# Cryptis

## Cryptographic Reasoning in Separation Logic

Arthur Azevedo de Amorim [1] Amal Ahmed [2] Marco Gaboardi [3]

May 22, 2023

[1]Rochester Institute of Technology

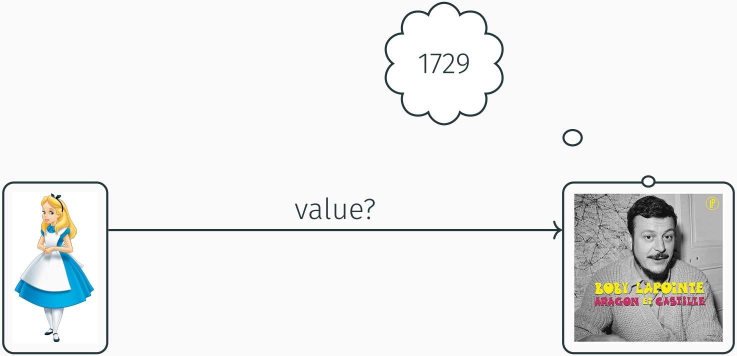
[2]Northeastern University

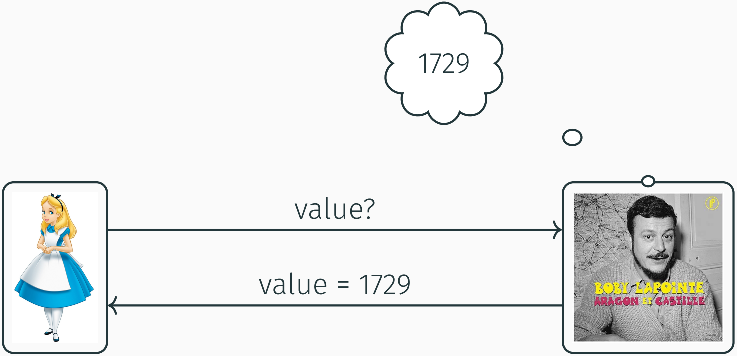
[3]Boston University

Let us connect to the cloud!

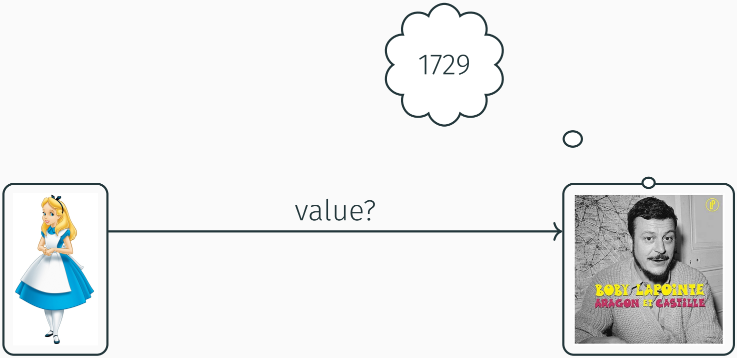

BOBY LAPOINTE
ARAGON ET CASTILLE

1729


1729
value?
BOBY LAPOINTE
ARAGON ET CASTILLE

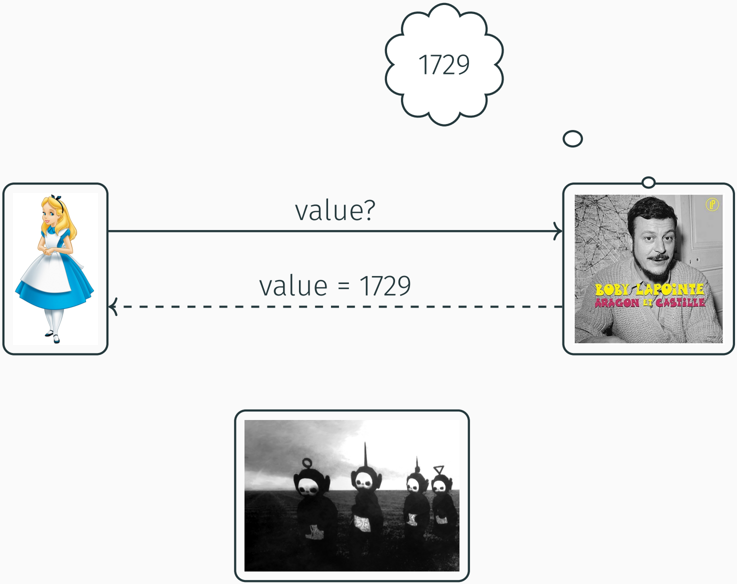

1729
value?
value = 1729
BOBY LAPOINTE
ARAGON ET CASTILLE

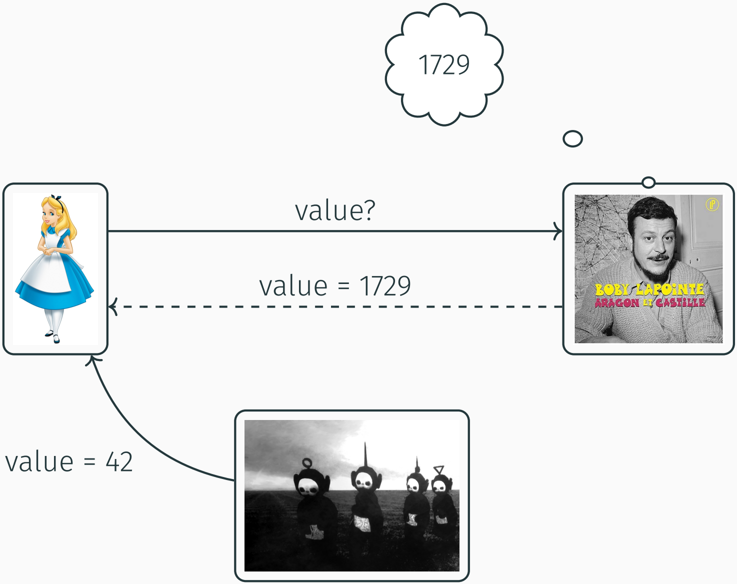
Can Alice trust this value?


1729
value?
BOBY LAPOINTE
ARAGON ET CASTILLE

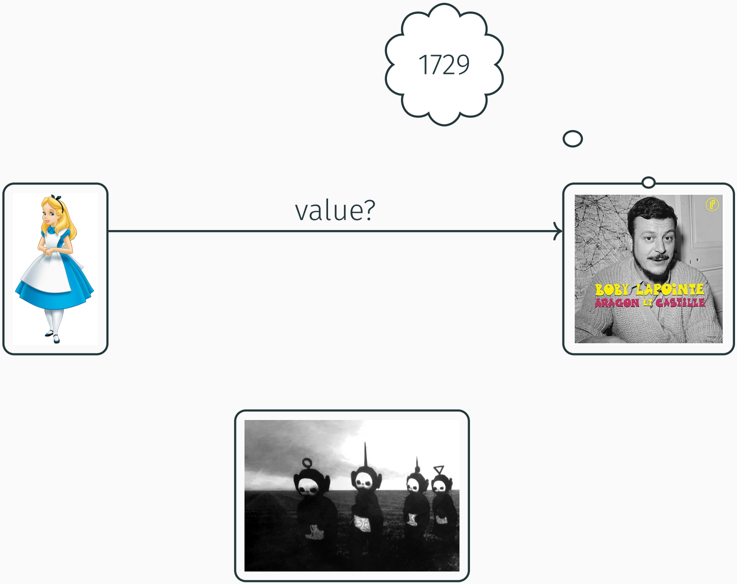
1729

value?



1729
value?
value = 1729
BOBY LAPOINTE
ARAGON ET CASTILLE


1729
value?
BOBY LAPOINTE
ARAGON ET CASTILLE
value = 1729
value = 42

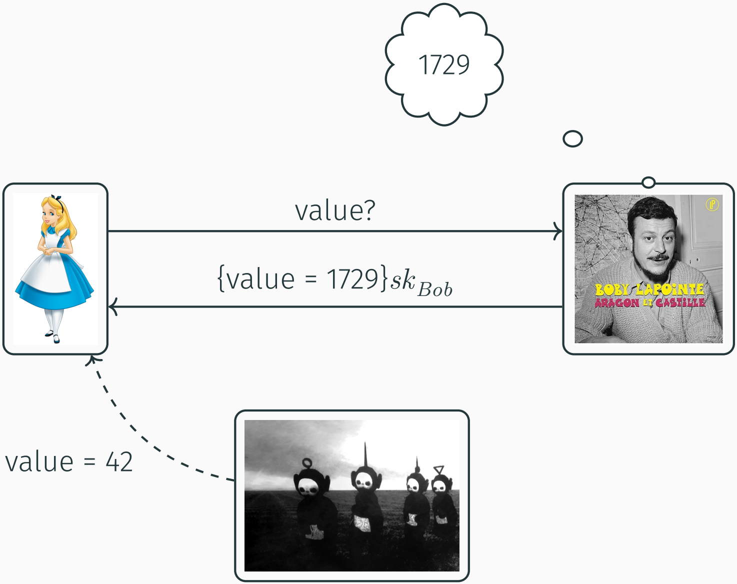
Possible Solution:
*Digital Signatures*


1729
value?
BOBY LAPOINTE
ARAGON ET CASTILLE

1729

value?

$\{\text{value = 1729}\}sk_{Bob}$



1729
value?
{value = 1729}$sk_{Bob}$
BOBY LAPOINTE
ARAGON ET CASTILLE
value = 42

# Why does this work?

Invariant reasoning

- Only Bob has his signing key

# Why does this work?

Invariant reasoning

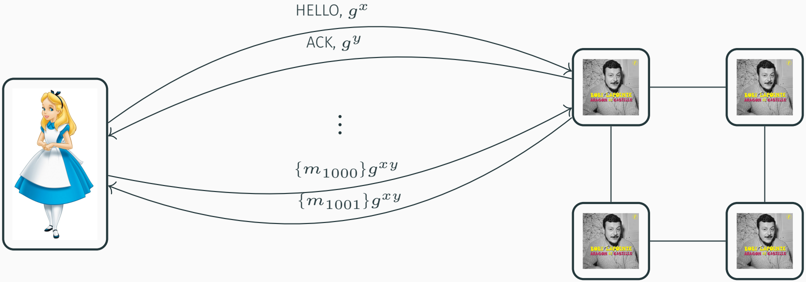
- Only Bob has his signing key
- Bob promises only to sign values stored in his server

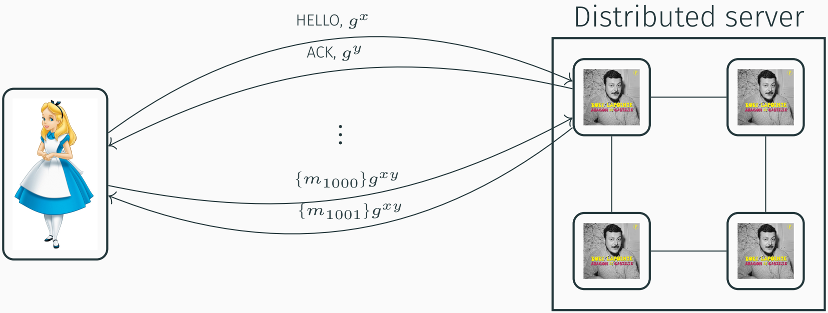
# Why does this work?

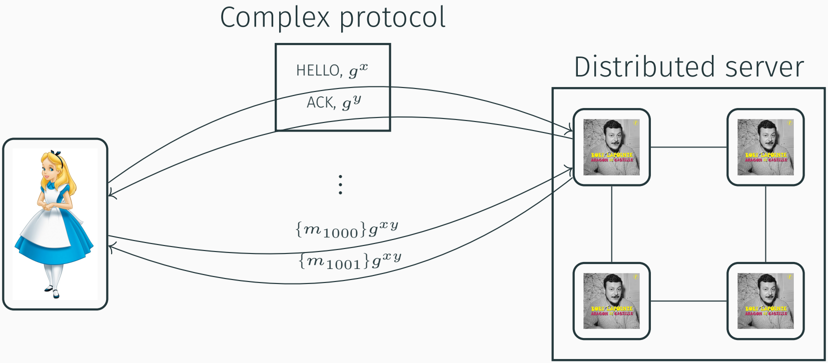
Invariant reasoning

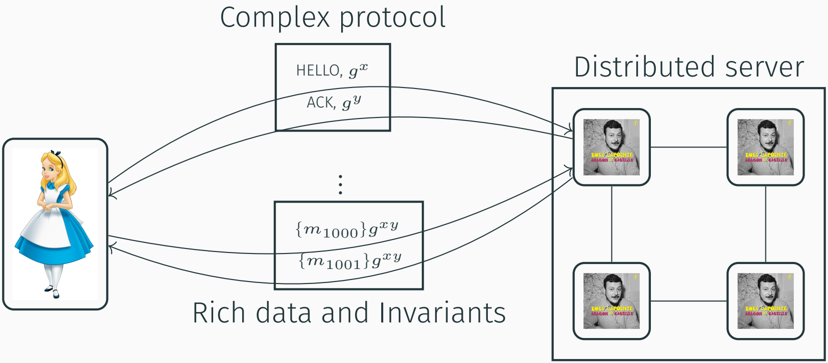
- Only Bob has his signing key
- Bob promises only to sign values stored in his server
- When Alice verifies the signature, she concludes Bob must have the value

How can this scale to *complex* systems with *cryptographic components*?


HELLO, $g^x$
ACK, $g^y$
$\vdots$
$\{m_{1000}\}g^{xy}$
$\{m_{1001}\}g^{xy}$


Distributed server
HELLO, $g^x$
ACK, $g^y$
⋮
$\{m_{1000}\}g^{xy}$
$\{m_{1001}\}g^{xy}$


Complex protocol
HELLO, $g^x$
ACK, $g^y$
$\vdots$
$\{m_{1000}\}g^{xy}$
$\{m_{1001}\}g^{xy}$
Distributed server

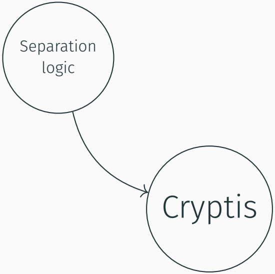

Complex protocol
HELLO, $g^x$
ACK, $g^y$
$\vdots$
$\{m_{1000}\}g^{xy}$
$\{m_{1001}\}g^{xy}$
Rich data and Invariants
Distributed server

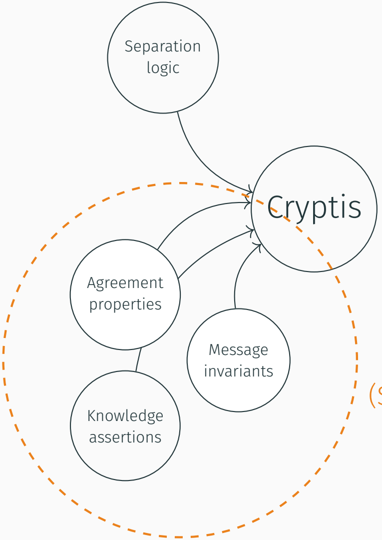
# Cryptis

## Cryptographic Reasoning in Separation Logic

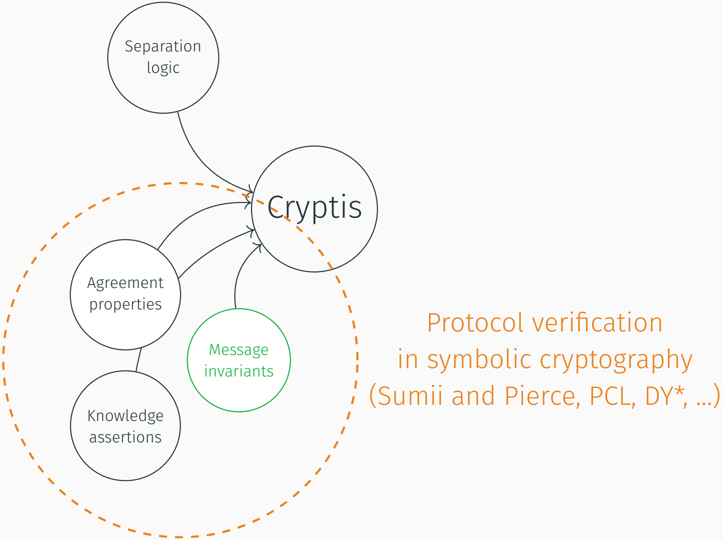
`https://github.com/arthuraa/cryptis`

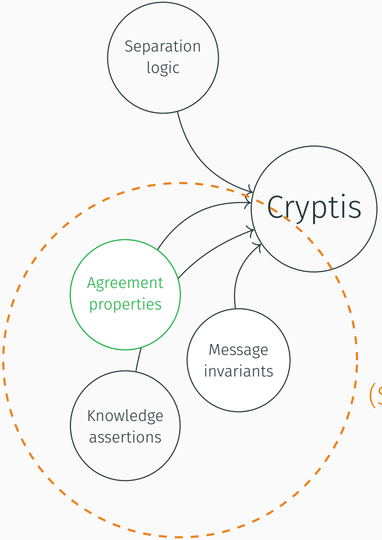
Cryptis


Separation logic
Cryptis

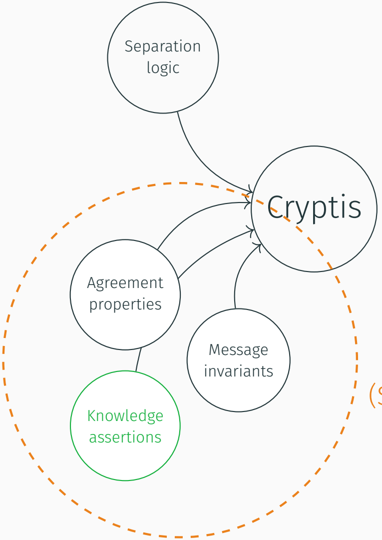



Protocol verification
in symbolic cryptography
(Sumii and Pierce, PCL, DY*, …)

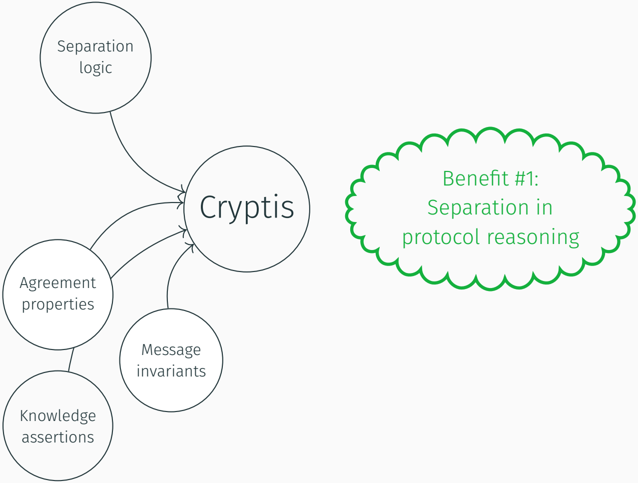

Separation logic
Cryptis
Agreement properties
Message invariants
Knowledge assertions
Protocol verification
in symbolic cryptography
(Sumii and Pierce, PCL, DY*, …)

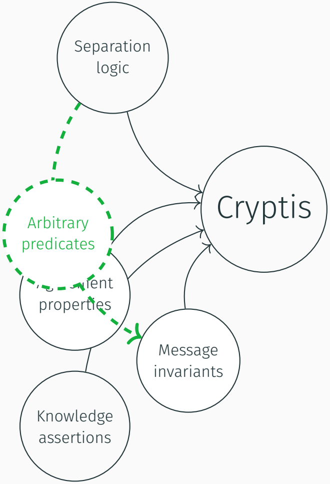



Protocol verification
in symbolic cryptography
(Sumii and Pierce, PCL, DY*, …)

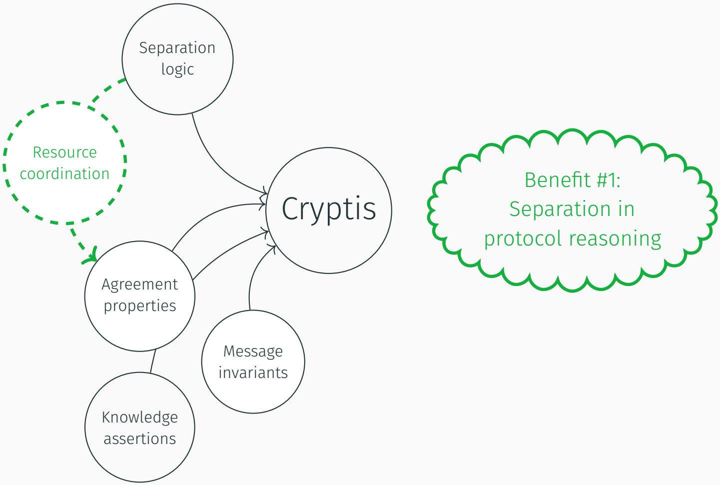


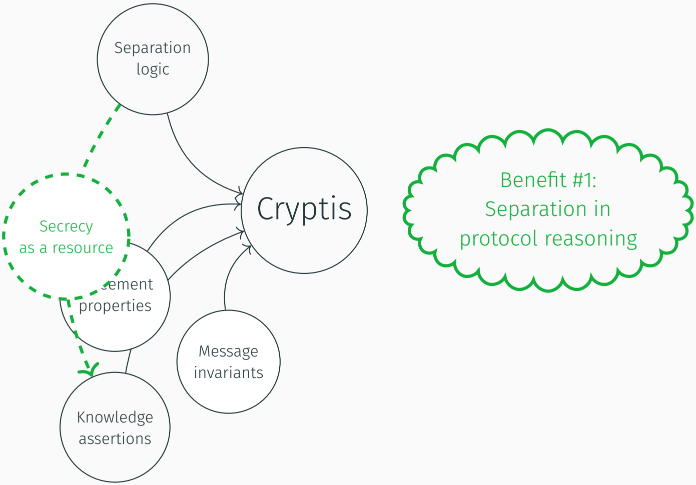

Protocol verification
in symbolic cryptography
(Sumii and Pierce, PCL, DY*, …)

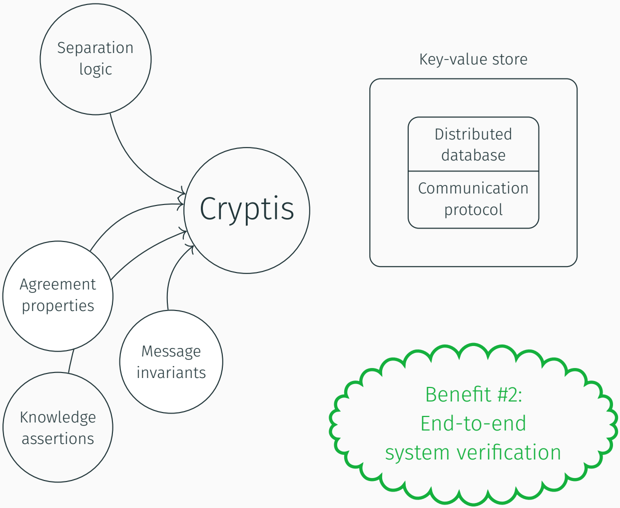

Separation logic
Cryptis
Agreement properties
Message invariants
Knowledge assertions
Benefit #1: Separation in protocol reasoning

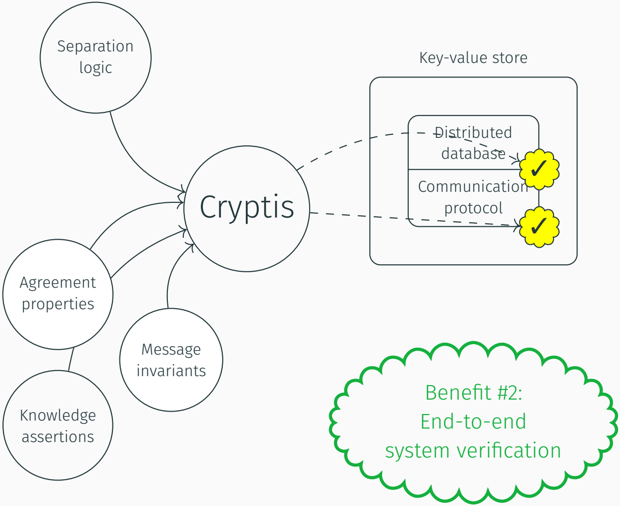


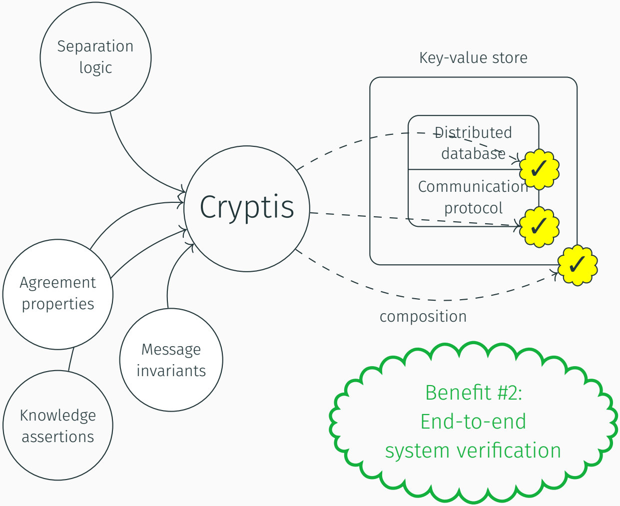

Benefit #1:
Separation in
protocol reasoning


Separation logic
Resource coordination
Cryptis
Agreement properties
Message invariants
Knowledge assertions
Benefit #1: Separation in protocol reasoning


Separation logic
Secrecy as a resource
ement properties
Knowledge assertions
Message invariants
Cryptis
Benefit #1:
Separation in
protocol reasoning


Separation logic
Cryptis
Agreement properties
Message invariants
Knowledge assertions
Key-value store
Distributed database
Communication protocol
Benefit #2:
End-to-end
system verification


Separation logic
Key-value store
Distributed database
Communication protocol
Cryptis
Agreement properties
Message invariants
Knowledge assertions
Benefit #2:
End-to-end
system verification


Separation logic
Key-value store
Distributed database
Communication protocol
Cryptis
Agreement properties
Message invariants
Knowledge assertions
composition
Benefit #2:
End-to-end
system verification

# A Tour of Cryptis

# The Needham-Schroeder-Lowe protocol

# The Needham-Schroeder-Lowe protocol

$$\{\texttt{"m1"}, n_A, pk_A\} @ pk_B$$

# The Needham-Schroeder-Lowe protocol

$$\{\texttt{"m1"}, n_A, pk_A\} @\, pk_B$$

$$\{\texttt{"m2"}, n_A, n_B, pk_B\} @\, pk_A$$

# The Needham-Schroeder-Lowe protocol

$$\{\texttt{"m1"}, n_A, pk_A\}@\,pk_B$$

$$\{\texttt{"m2"}, n_A, n_B, pk_B\}@\,pk_A$$

$$\{\texttt{"m3"}, n_B\}@\,pk_B$$

# The Needham-Schroeder-Lowe protocol

$$\{\texttt{"m1"}, n_A, pk_A\}@\,pk_B$$

$$\{\texttt{"m2"}, n_A, n_B, pk_B\}@\,pk_A$$

$$\{\texttt{"m3"}, n_B\}@\,pk_B$$


Goals:

# The Needham-Schroeder-Lowe protocol

$\{\texttt{"m1"}, n_A, pk_A\}@\,pk_B$

$\{\texttt{"m2"}, n_A, n_B, pk_B\}@\,pk_A$

$\{\texttt{"m3"}, n_B\}@\,pk_B$


Goals:

- $n_A$ and $n_B$ are secret

# The Needham-Schroeder-Lowe protocol

$\{\texttt{"m1"}, n_A, pk_A\}@\,pk_B$

$\{\texttt{"m2"}, n_A, n_B, pk_B\}@\,pk_A$

$\{\texttt{"m3"}, n_B\}@\,pk_B$


Goals:

- $n_A$ and $n_B$ are secret
- agents agree on their identities

# The Needham-Schroeder-Lowe protocol

$\{\texttt{"m1"}, n_A, pk_A\}@\, pk_B$

$\{\texttt{"m2"}, n_A, n_B, pk_B\}@\, pk_A$

$\{\texttt{"m3"}, n_B\}@\, pk_B$


Goals:

- $n_A$ and $n_B$ are secret
- agents agree on their identities
- ...and can exchange resources

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\text{public}(pk_A) \quad \Box\neg\,\text{public}(sk_A)$

$\text{public}(pk_B) \quad \Box\neg\,\text{public}(sk_B)$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathsf{public}(pk_A)$ $\quad \Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\quad \Box\neg\,\mathsf{public}(sk_B)$

Can travel through network

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathsf{public}(pk_A)$ $\Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\Box\neg\,\mathsf{public}(sk_B)$

Cannot travel through network

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathsf{public}(pk_A)$ $\quad \Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\quad \Box\neg\,\mathsf{public}(sk_B)$

$\Box\neg\,\mathsf{public}(n_A)$ $\quad \mathsf{fresh}(n_A)$

$$\{\top\}\ \mathsf{new\_nonce}()\ \left\{ n.\ \begin{array}{l} \Box\neg\,\mathsf{public}(n) \\ *\,\mathsf{fresh}(n) \end{array} \right\}$$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\text{public}(pk_A)$ $\quad \Box\neg\,\text{public}(sk_A)$

$\text{public}(pk_B)$ $\quad \Box\neg\,\text{public}(sk_B)$

$\Box\neg\,\text{public}(n_A)$ $\quad \text{fresh}(n_A)$

$\text{begin}(pk_A, pk_B, n_A)$

Protocol specific

$$\text{fresh}(n_A) \Rrightarrow \text{begin}(pk_A, pk_B, n_A)$$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\text{public}(pk_A)$ $\quad \Box\neg\,\text{public}(sk_A)$

$\text{public}(pk_B)$ $\quad \Box\neg\,\text{public}(sk_B)$

$\Box\neg\,\text{public}(n_A)$ $\quad \text{fresh}(n_A)$

$\text{begin}(pk_A, pk_B, n_A)$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathsf{public}(pk_A)$ $\Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\Box\neg\,\mathsf{public}(sk_B)$

$\Box\neg\,\mathsf{public}(n_A)$ $\mathsf{fresh}(n_A)$

$\mathsf{begin}(pk_A, pk_B, n_A)$

$$\frac{\Box\neg\,\mathsf{public}(sk) \qquad I(t, pk)}{\mathsf{public}(\{t\}@\,pk)}$$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathrm{public}(pk_A)$ $\quad \Box\neg\,\mathrm{public}(sk_A)$

$\mathrm{public}(pk_B)$ $\quad \Box\neg\,\mathrm{public}(sk_B)$

$\Box\neg\,\mathrm{public}(n_A)$ $\quad \mathrm{fresh}(n_A)$

$\mathrm{begin}(pk_A, pk_B, n_A)$

Protocol specific

$$\frac{\Box\neg\,\mathrm{public}(sk) \qquad I(t, pk)}{\mathrm{public}(\{t\}@\,pk)}$$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Protocol specific

$$\frac{\Box\neg\,\mathsf{public}(sk) \qquad I(t, pk)}{\mathsf{public}(\{t\}@\,pk)}$$

Proof state

$\mathsf{public}(pk_A)$ $\Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\Box\neg\,\mathsf{public}(sk_B)$

$\Box\neg\,\mathsf{public}(n_A)$ $\mathsf{fresh}(n_A)$

$\mathsf{begin}(pk_A, pk_B, n_A)$

$\mathsf{public}(m_1)$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathsf{public}(pk_A)$ $\Box\neg\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\Box\neg\mathsf{public}(sk_B)$

$\Box\neg\mathsf{public}(n_A)$ $\mathsf{fresh}(n_A)$

$\mathsf{begin}(pk_A, pk_B, n_A)$

$\mathsf{public}(m_1)$

$\{\mathsf{public}(m)\}\ \mathsf{send}(m)\ \{\top\}$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Proof state

$\mathsf{public}(pk_A)$ $\quad \Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\quad \Box\neg\,\mathsf{public}(sk_B)$

$\Box\neg\,\mathsf{public}(n_A)$ $\quad \mathsf{fresh}(n_A)$

$\mathsf{begin}(pk_A, pk_B, n_A)$

$\mathsf{public}(m_1)$ $\quad \mathsf{public}(m_2)$

$\{\top\}\ \mathsf{recv}(m)\ \{m.\ \mathsf{public}(m)\}$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

$$\frac{\text{public}(\{t\}@\,pk)}{I(t, pk)}$$

Proof state

$\text{public}(pk_A)$ $\quad \Box\neg\,\text{public}(sk_A)$

$\text{public}(pk_B)$ $\quad \Box\neg\,\text{public}(sk_B)$

$\Box\neg\,\text{public}(n_A)$ $\quad \text{fresh}(n_A)$

$\text{begin}(pk_A, pk_B, n_A)$

$\text{public}(m_1)$ $\quad \text{public}(m_2)$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Protocol specific

$$\frac{\mathsf{public}(\{t\}@\,pk)}{I(t, pk)}$$

Proof state

$\mathsf{public}(pk_A)$ $\quad \Box\neg\,\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\quad \Box\neg\,\mathsf{public}(sk_B)$

$\Box\neg\,\mathsf{public}(n_A)$ $\quad \mathsf{fresh}(n_A)$

$\mathsf{begin}(pk_A, pk_B, n_A)$

$\mathsf{public}(m_1)$ $\quad \mathsf{public}(m_2)$

$\Box\neg\,\mathsf{public}(n_B)$

$\mathsf{accept}(pk_A, pk_B, n_A, n_B)$

# Initiator proof

```
let nA = new_nonce() in
let m1 = {"m1", nA, pkA}@pkB in
send m1;
let m2 = recv () in
let {="m2", =nA, nB, =pkB} = dec m2 skA in
(* ... *)
```

Protocol specific

$$\mathsf{begin}(pk_A, pk_B, n_A) * \mathsf{accept}(pk_A, pk_B, n_A, n_B) \Rrightarrow P$$

Proof state

$\mathsf{public}(pk_A)$ $\quad \Box\neg\mathsf{public}(sk_A)$

$\mathsf{public}(pk_B)$ $\quad \Box\neg\mathsf{public}(sk_B)$

$\Box\neg\mathsf{public}(n_A)$ $\quad \mathsf{fresh}(n_A)$

$\mathsf{begin}(pk_A, pk_B, n_A)$

$\mathsf{public}(m_1)$ $\quad \mathsf{public}(m_2)$

$\Box\neg\mathsf{public}(n_B)$

$\mathsf{accept}(pk_A, pk_B, n_A, n_B)$

$P$

Application specific; provided by responder

# Using the Protocol

# A key-value store (client interface)

$$\{\top\}\ \mathsf{create}(k, v)\ \left\{b.\ b = 1 \Rightarrow k \mapsto^{\mathsf{db}} v\right\}$$

$$\left\{k \mapsto^{\mathsf{db}} v\right\}\ \mathsf{get}(k)\ \left\{v'.\ k \mapsto^{\mathsf{db}} v * v' = v\right\}$$

$$\left\{k \mapsto^{\mathsf{db}} v'\right\}\ \mathsf{set}(k, v)\ \left\{k \mapsto^{\mathsf{db}} v\right\}$$

# A key-value store (client interface)

$$\{\top\}\ \mathsf{create}(k, v)\ \left\{b.\ b = 1 \Rightarrow k \mapsto^{\mathsf{db}} v\right\}$$

$$\left\{k \mapsto^{\mathsf{db}} v\right\}\ \mathsf{get}(k)\ \left\{v'.\ k \mapsto^{\mathsf{db}} v * v' = v\right\}$$

$$\left\{k \mapsto^{\mathsf{db}} v'\right\}\ \mathsf{set}(k, v)\ \left\{k \mapsto^{\mathsf{db}} v\right\}$$

# Remote points-to

$$k \mapsto^{\mathsf{db}} v \triangleq \exists n : \mathbb{N}, l_c \mapsto n * \mathsf{clientView}(k, v, n)$$

$$k \mapsto^{\mathsf{db}}_{\mathsf{server}} v \triangleq \exists n : \mathbb{N}, \mathsf{stored}(k, n, v) * \mathsf{serverView}(k, v, n)$$

# Remote points-to

$$k \mapsto^{\mathsf{db}} v \triangleq \exists n : \mathbb{N}, l_c \mapsto n * \mathsf{clientView}(k, v, n)$$

$$k \mapsto^{\mathsf{db}}_{\mathsf{server}} v \triangleq \exists n : \mathbb{N}, \mathsf{stored}(k, n, v) * \mathsf{serverView}(k, v, n)$$

Key $k$ maps to $v$,
and the current time is $n$

# Remote points-to

$$k \mapsto^{\mathsf{db}} v \triangleq \exists n : \mathbb{N}, l_c \mapsto n * \mathsf{clientView}(k, v, n)$$

$$k \mapsto^{\mathsf{db}}_{\mathsf{server}} v \triangleq \exists n : \mathbb{N}, \mathsf{stored}(k, n, v) * \mathsf{serverView}(k, v, n)$$

Local state of server

# Remote points-to

$$k \mapsto^{\mathsf{db}} v \triangleq \exists n : \mathbb{N}, l_c \mapsto n * \mathsf{clientView}(k, v, n)$$

$$k \mapsto^{\mathsf{db}}_{\mathsf{server}} v \triangleq \exists n : \mathbb{N}, \mathsf{stored}(k, n, v) * \mathsf{serverView}(k, v, n)$$

Key $k$ mapped to $v$ at time $n$

# View laws

$$
\begin{aligned}
\mathsf{clientView}(k, v, n) * \mathsf{serverView}(k, v', m) &\twoheadrightarrow n \geq m \\
\mathsf{clientView}(k, v, n) * \mathsf{serverView}(k, v', n) &\twoheadrightarrow v = v' \\
\mathsf{serverView}(k, v, n) &\twoheadrightarrow \Box\mathsf{serverView}(k, v, n) \\
\mathsf{clientView}(k, v, n) &\Rrightarrow \mathsf{clientView}(k, v', n+1) \\
\mathsf{clientView}(k, v, n) &\twoheadrightarrow \mathsf{serverView}(k, v, n)
\end{aligned}
$$

# Server message invariants

$$\frac{\text{serverView}(k, v, n)}{\text{public}(\{\texttt{"set\_req"}, k, v, n\}@\,pk)}$$

$$\frac{\text{serverView}(k, v, n)}{\text{public}(\{\texttt{"get\_resp"}, k, v, n\}@\,pk)}$$

# Wrapping Up

# Conclusion

- Cryptis integrates cryptographic reasoning with system verification

# Conclusion

- Cryptis integrates cryptographic reasoning with system verification
- Other features: Diffie-Hellman operations, analysis of key compromise scenarios, forward secrecy, ...

# Conclusion

- Cryptis integrates cryptographic reasoning with system verification
- Other features: Diffie-Hellman operations, analysis of key compromise scenarios, forward secrecy, ...
- Future work: Generate models from implementations, relational analysis of secrecy

Thank you!