Cryptographic proofs in Approxis

Zacharie Moughanim supervised by Lars Birkedal and Phillip Haselwarter

2025-08-27

AARHUS
UNIVERSITET
INSTITUT FOR DATALOGI

What is a (cryptographic) protocol ?

Cryptographic primitives: encryption, hash function, signature...

What is a (cryptographic) protocol ?

Cryptographic primitives: encryption, hash function, signature...

Symmetric encryption: $\mathrm{senc}, \mathrm{sdec}$, key $k \twoheadleftarrow \mathrm{keygen}$

What is a (cryptographic) protocol ?

Cryptographic primitives: encryption, hash function, signature...

Symmetric encryption: $\mathrm{senc}, \mathrm{sdec}$, key $k \twoheadleftarrow \mathrm{keygen}$: $\mathrm{sdec}(k, \mathrm{senc}(k, m)) = m$

What is a (cryptographic) protocol ?

Cryptographic primitives: encryption, hash function, signature...

Symmetric encryption: $\mathrm{senc}, \mathrm{sdec}$, key $k \twoheadleftarrow \mathrm{keygen}$: $\mathrm{sdec}(k, \mathrm{senc}(k, m)) = m$

$\{m\}_k \equiv \mathrm{senc}(k, m)$

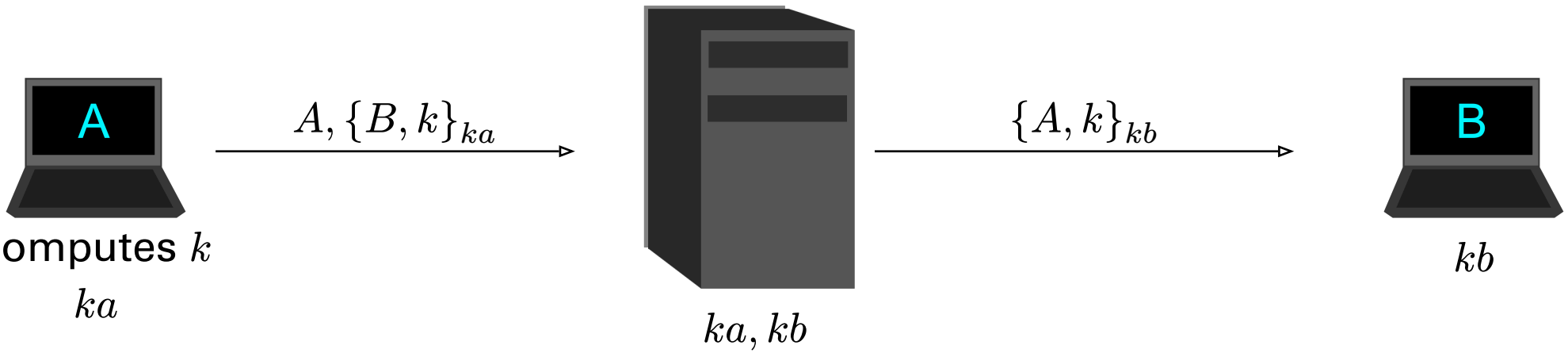
What is a (cryptographic) protocol ?

Cryptographic primitives: encryption, hash function, signature...

Symmetric encryption: $\mathrm{senc}, \mathrm{sdec}$, key $k \twoheadleftarrow \mathrm{keygen}$: $\mathrm{sdec}(k, \mathrm{senc}(k, m)) = m$

$\{m\}_k \equiv \mathrm{senc}(k, m)$

Wide-Mouthed Frog (WMF) protocol



Separation logic

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Separation logic

Goal: Reason about program with a heap (e.g. C or OCaml with references)

- Specifications as logical formulas;

Separation logic

Goal: Reason about program with a heap (e.g. C or OCaml with references)

- Specifications as logical formulas;
- points-to predicate $l \mapsto 42$;

Separation logic

Goal: Reason about program with a heap (e.g. C or OCaml with references)

- Specifications as logical formulas;
- points-to predicate $l \mapsto 42$;
- Hoare triple $\{l \mapsto x\} l := 42 \{l \mapsto 42\}$.

Iris examples (https://iris-project.org)

A separation logic, formalized in Coq/Rocq

Iris examples (https://iris-project.org)

A separation logic, formalized in Coq/Rocq

```
Definition update : val := λ: "x", "x" <- ! "x" + #1.
```

Iris examples (https://iris-project.org)

A separation logic, formalized in Coq/Rocq

```
Definition update : val := λ: "x", "x" <- ! "x" + #1.

Lemma update_spec (l : loc) (x : Z) :
  {{{ l ↦ #x }}} update #l {{{ RET #(); l ↦ #(x + 1) }}}.
```

Iris examples (https://iris-project.org)

A separation logic, formalized in Coq/Rocq

```
Definition update : val := λ: "x", "x" <- ! "x" + #1.

Lemma update_spec (l : loc) (x : Z) :
  {{{ l ↦ #x }}} update #l {{{ RET #(); l ↦ #(x + 1) }}}.
```

… Hoare triple ⇝ unary property

Cryptography: indistinguishability between 2 “sides”

Contextual equivalence

$e_1 \underset{\text{ctx}}{\cong} e_2$: **for any context** C (program with a hole $\square$), $C[e_1]$ terminates iff $C[e_2]$ terminates.

Contextual equivalence

$e_1 \underset{\text{ctx}}{\cong} e_2$: **for any context** C (program with a hole $\square$), $C[e_1]$ terminates iff $C[e_2]$ terminates.

Example

$e_1 \underset{\text{ctx}}{\cong} e_2$: int terminating.

Contextual equivalence

$e_1 \underset{\text{ctx}}{\cong} e_2$: **for any context** C (program with a hole $\square$), $C[e_1]$ terminates iff $C[e_2]$ terminates.

Example

$e_1 \underset{\text{ctx}}{\cong} e_2$: int terminating.

$\Longrightarrow$ They evaluate to the same integer.

Contextual equivalence

$e_1 \underset{\text{ctx}}{\cong} e_2$: **for any context** C (program with a hole $\square$), $C[e_1]$ terminates iff $C[e_2]$ terminates.

Example

$e_1 \underset{\text{ctx}}{\cong} e_2$: int terminating.

$\Longrightarrow$ They evaluate to the same integer.

$C_{42} := \text{if } \square = 42 \text{ then } () \text{ else loop}$

Contextual equivalence

$e_1 \underset{\text{ctx}}{\cong} e_2$: **for any context** C (program with a hole $\square$), $C[e_1]$ terminates iff $C[e_2]$ terminates.

Example

$e_1 \underset{\text{ctx}}{\cong} e_2$: int terminating.

$\Rightarrow$ They evaluate to the same integer.

$C_{42} := \text{if } \square = 42 \text{ then } () \text{ else loop}$

$C_k := \text{if } \square = k \text{ then } () \text{ else loop}$

... Hard to prove e.g. by induction on the context

$\rightsquigarrow$ we prove logical refinements: $e \underset{\log}{\ll} e' : T$

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs: rand $N \rightsquigarrow \begin{cases} 0 \\ 1 \\ \vdots \\ N \end{cases}$ with equiprobability

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs: $\text{rand}\ N \rightsquigarrow \begin{cases} 0 \\ 1 \\ \vdots \\ N \end{cases}$ with equiprobability

$\text{rand}\ 3 \underset{\log}{\ll} \text{rand}\ 1 + 2 * \text{rand}\ 1$

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs: $\text{rand}\ N \rightsquigarrow \begin{cases} 0 \\ 1 \\ \vdots \\ N \end{cases}$ with equiprobability

$\text{rand}\ 3 \underset{\log}{\ll} \text{rand}\ 1 + 2 * \text{rand}\ 1$

Error credits: $↯\ \varepsilon$

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs: $\text{rand } N \rightsquigarrow \begin{cases} 0 \\ 1 \\ \vdots \\ N \end{cases}$ with equiprobability

$\text{rand } 3 \underset{\log}{\ll} \text{rand } 1 + 2 * \text{rand } 1$

Error credits: $↯\ \varepsilon$

$↯\ \frac{1}{N+1} \vdash \text{rand } N \underset{\log}{\ll} \text{let } x = \text{rand } N \text{ in if } x = 0 \text{ then } -1 \text{ else } x$

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs: $\mathtt{rand}\ N \rightsquigarrow \begin{cases} 0 \\ 1 \\ \vdots \\ N \end{cases}$ with equiprobability

$\mathtt{rand}\ 3 \underset{\log}{\ll} \mathtt{rand}\ 1 + 2 * \mathtt{rand}\ 1$

Error credits: $\lightning\ \varepsilon$

$\lightning\ \frac{1}{N+1} \vdash \mathtt{rand}\ N \underset{\log}{\ll} \mathtt{let}\ x = \mathtt{rand}\ N\ \mathtt{in}\ \mathtt{if}\ x = 0\ \mathtt{then} - 1\ \mathtt{else}\ x$

- Approxis logic

Iris + Probabilistic programs + Logical relations + Error credits = Approxis

Clutch: Iris + probabilistic programs: $\mathsf{rand}\ N \rightsquigarrow \begin{cases} 0 \\ 1 \\ \vdots \\ N \end{cases}$ with equiprobability

$\mathsf{rand}\ 3 \underset{\log}{\ll} \mathsf{rand}\ 1 + 2 * \mathsf{rand}\ 1$

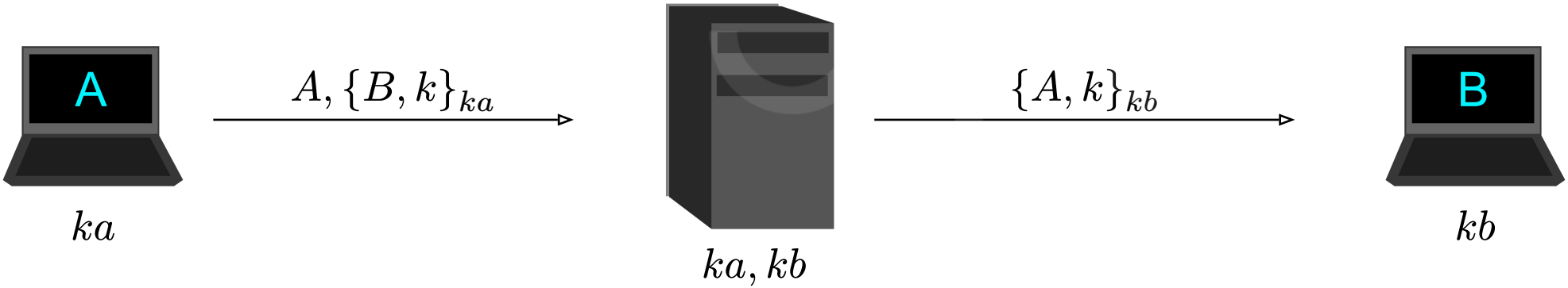
Error credits: $↯\ \varepsilon$

$↯\ \frac{1}{N+1} \vdash \mathsf{rand}\ N \underset{\log}{\ll} \mathsf{let}\ x = \mathsf{rand}\ N\ \mathsf{in}\ \mathsf{if}\ x = 0\ \mathsf{then} - 1\ \mathsf{else}\ x$

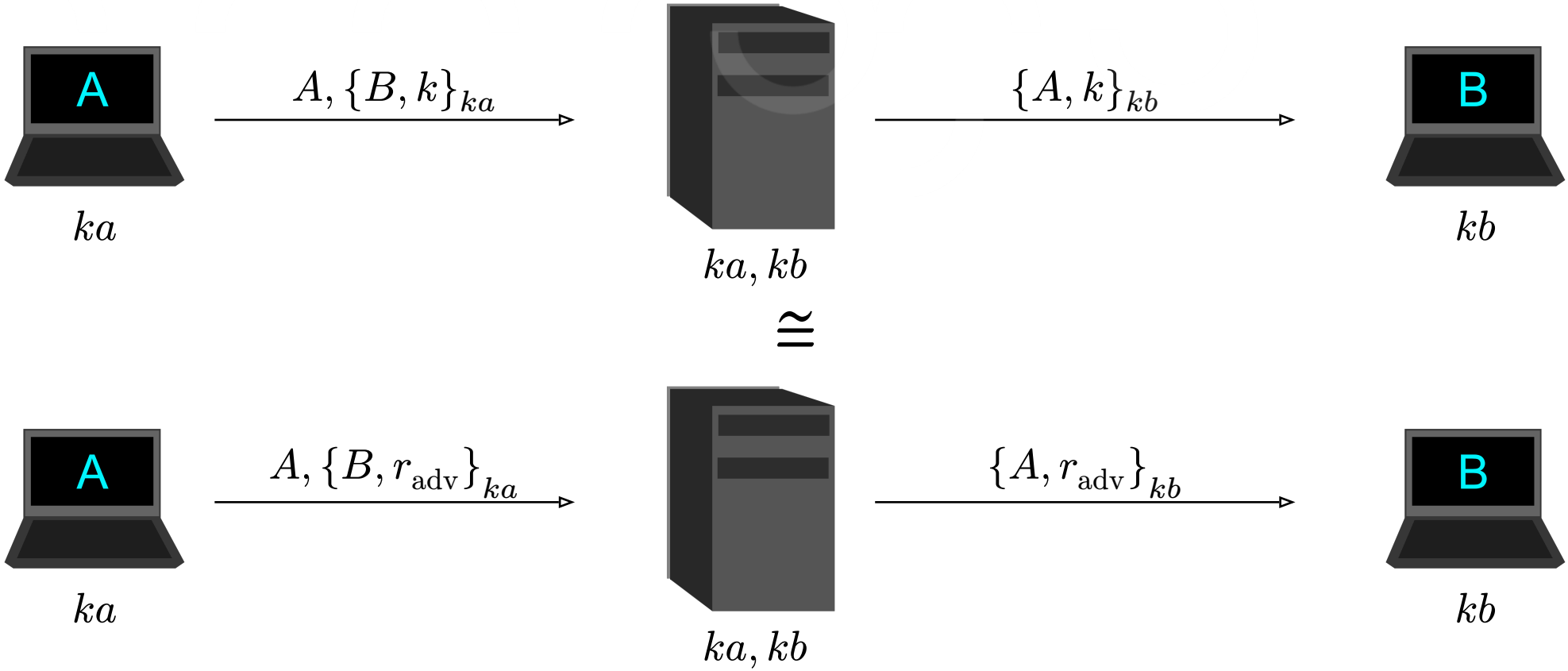
- Approxis logic

Computational security: properties up to a negligible error

What is a secure protocol ?



What is a secure protocol ?



A protocol as a program

Eavesdropping attacker

Active attacker

A protocol as a program

Eavesdropping attacker

WMF eav

$ka, kb :=_{\$} \text{keygen}$

$k :=_{\$} \{0,1\}^{\eta}$

// First message

let msg1 $= (A, \text{senc}(ka, (B, k)))$ in

// Second message

let mdec $= \text{sdec}(\text{snd}\,\text{m}, ka)$ in

let sender $=$ fst m in

$(\text{msg1}, \text{senc}(kb, (\text{sender}, \text{snd}\,\text{msg})))$

Active attacker

A protocol as a program

Eavesdropping attacker

WMF eav

$ka, kb :=_{\$}$ keygen

$k :=_{\$} \{0,1\}^{\eta}$

// First message

let msg1 $= (A, \mathrm{senc}(ka, (B, k)))$ in

// Second message

let mdec $= \mathrm{sdec}(\mathrm{snd}\ \mathsf{m}, ka)$ in

let sender $=$ fst m in

$(\mathsf{msg1}, \mathrm{senc}(kb, (\mathsf{sender}, \mathrm{snd}\ \mathsf{msg})))$

Active attacker

WMF

$ka, kb :=_{\$}$ keygen

FirstMessageA():

  $k :=_{\$} \{0,1\}^{\eta}$

  return $(A, \mathrm{senc}(ka, (B, k)))$

SecondMessageS(m):

  msg := $\mathrm{sdec}(\mathrm{snd}\ \mathsf{m}, ka)$

  sender := fst m

  return $\mathrm{senc}(kb, (\mathsf{sender}, \mathrm{snd}\ \mathsf{msg}))$

Assumption on the encryption scheme

$\mathcal{L}^{\Sigma}_{\text{real}}$	$\mathcal{L}^{\Sigma}_{\text{rand}}$
$k :=_{\$} \Sigma.\text{keygen}$	ENCRYPT(m):
ENCRYPT(m):	$r :=_{\$} \{0,1\}^{\eta}$
return $\Sigma.\text{senc}(k, m)$	return r

IND-CPA$ game for symmetric encryption

Assumption on the encryption scheme

$\mathcal{L}^{\Sigma}_{\text{real}}$	$\mathcal{L}^{\Sigma}_{\text{rand}}$

$k :=_{\$} \Sigma.\text{keygen}$
ENCRYPT(m):
  return $\Sigma.\text{senc}(k, m)$
	ENCRYPT(m):
	  $r :=_{\$} \{0,1\}^{\eta}$
	  return r

IND-CPA$ game for symmetric encryption

$(\text{senc}, \text{sdec}, \text{keygen})$ satisfies IND-CPA$ if $\mathcal{L}^{\Sigma}_{\text{real}}$ and $\mathcal{L}^{\Sigma}_{\text{rand}}$ are indistinguishable

Proof

WMF

$ka, kb :=_{\$} \text{keygen}$

FirstMessageA():

$k :=_{\$} \{0,1\}^{\eta}$

send $(A, \text{senc}(ka, (B, k)))$

SecondMessageS(m):

msg $:=$ sdec(snd m, ka)

sender $:=$ fst m

send senc(kb, (sender, snd msg))

Proof

WMF

$ka, kb :=_{\$} \text{keygen}$

FirstMessageA():

$k :=_{\$} \{0,1\}^{\eta}$

send $(A, \text{senc}(ka, (B, k)))$

SecondMessageS(m):

msg $:=$ sdec(snd m, ka)

sender $:=$ fst m

send senc(kb, (sender, snd msg))

Proof

WMF

$ka, kb :=_{\$}$ keygen

FirstMessageA():

  $k :=_{\$} \{0,1\}^{\eta}$

  send $(A, \text{Encrypt}((A, k)))$

SecondMessageS(m):

  msg := sdec(snd m, ka)

  sender := fst m

  send senc(kb, (sender, snd msg))

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{real}}$

$ka :=_{\$}$ keygen

Encrypt(m):

  return Σ.senc(ka, m)

Proof

WMF

$ka, kb :=_{\$}$ keygen

FirstMessageA():

  $k :=_{\$} \{0,1\}^{\eta}$

  send $(A, \text{Encrypt}((A, k)))$

SecondMessageS(m):

  msg := sdec(snd m, ka)

  sender := fst m

  send senc(kb, (sender, snd msg))

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{rand}}$

Encrypt(m):

  $r :=_{\$} \{0,1\}^{\eta}$

  return r

Proof

WMF

$r :=_{\$} \{0,1\}^{2\eta}, ka, kb :=_{\$}$ keygen

FirstMessageA():

$k :=_{\$} \{0,1\}^{\eta}$

send (A, r)

SecondMessageS(m):

msg := sdec(snd m, ka)

sender := fst m

send senc(kb, (sender, snd msg))

Proof

WMF

$r_{\text{adv}} :=_{\$} \{0,1\}^{\eta}, ka, kb :=_{\$}$ keygen

FirstMessageA():

  $k :=_{\$} \{0,1\}^{\eta}$

  send $(A, \text{Encrypt}((B, r_{\text{adv}})))$

SecondMessageS(m):

  msg := sdec(snd m, ka)

  sender := fst m

  send senc(kb, (sender, snd msg))

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{rand}}$

Encrypt(m):

  $r :=_{\$} \{0,1\}^{\eta}$

  return r

Proof

WMF

$r_{\text{adv}} :=_{\$} \{0,1\}^{\eta}, ka, kb :=_{\$}$ keygen

FirstMessageA():

| $k :=_{\$} \{0,1\}^{\eta}$
| send $(A, \text{Encrypt}((B, r_{\text{adv}})))$

SecondMessageS(m):

| msg := sdec(snd m, ka)
| sender := fst m
| send senc(kb, (sender, snd msg))

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{real}}$

$ka :=_{\$}$ keygen

Encrypt(m):

| return Σ.senc(ka, m)

Proof

WMF

$r_{\text{adv}} :=_{\$} \{0,1\}^{\eta}, ka, kb :=_{\$} \text{keygen}$

FirstMessageA():

$k :=_{\$} \{0,1\}^{\eta}$

send $(A, \text{senc}(ka, (B, r_{\text{adv}})))$

SecondMessageS(m):

msg := sdec(snd m, ka)

sender := fst m

send senc(kb, (sender, snd msg))

Proof

WMF

$r_{\text{adv}} :=_{\$} \{0,1\}^{\eta}, ka, kb :=_{\$} \text{keygen}$

FirstMessageA():

$k :=_{\$} \{0,1\}^{\eta}$

send $(A, \text{senc}(ka, (B, r_{\text{adv}})))$

SecondMessageS(m):

msg := sdec(snd m, ka)

sender := fst m

send $\text{senc}(kb, (\text{sender}, \text{snd msg}))$

Proof

```
Lemma wmf_eav_delay_true__wmf_eav_true :
  ⊢ REL init_scheme (wmf_eav_delay #true #true) <<
      init_scheme (wmf_eav #true):
(lrel_rand * ((lrel_id * lrel_output) * (lrel_output * ()))).
```

- Eavesdropping attacker: proof complete ✓
- Active attacker: proof incomplete ✗ require more intricate games

Future works

- Add more games for more stronger assumptions;

Future works

- Add more games for more stronger assumptions;
- finish active attacker protocol;

Future works

- Add more games for more stronger assumptions;
- finish active attacker protocol;
- add primitives to make proofs simpler;

Future works

- Add more games for more stronger assumptions;
- finish active attacker protocol;
- add primitives to make proofs simpler;
- consider only polynomial contexts (contextual equivalence: too strong).

Conclusion

- Protocols implemented in Approxis;
  - proposed modelisation of protocols;
  - proved security for an example;

Conclusion

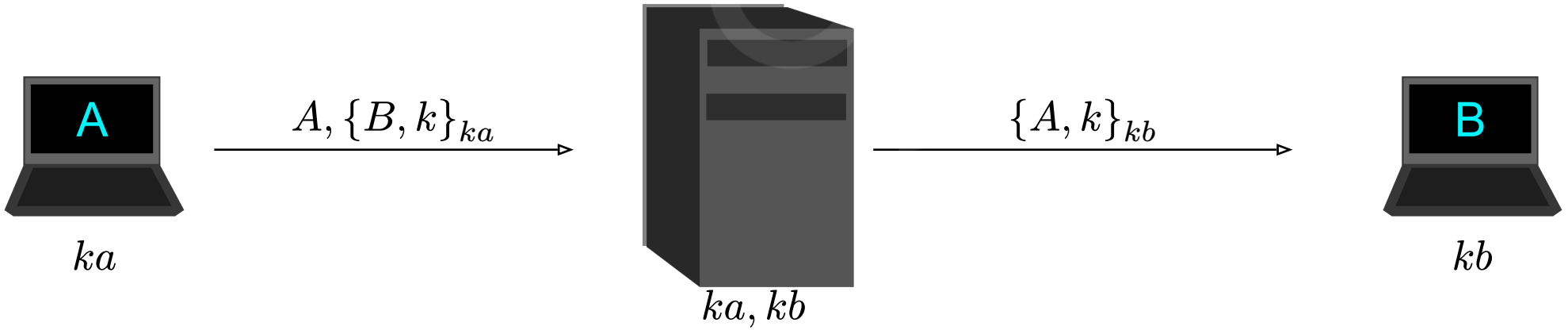
- Protocols implemented in Approxis;
  - proposed modelisation of protocols;
  - proved security for an example;
- implemented new cryptographic game: IND-CCA, INT-PTXT;

Conclusion

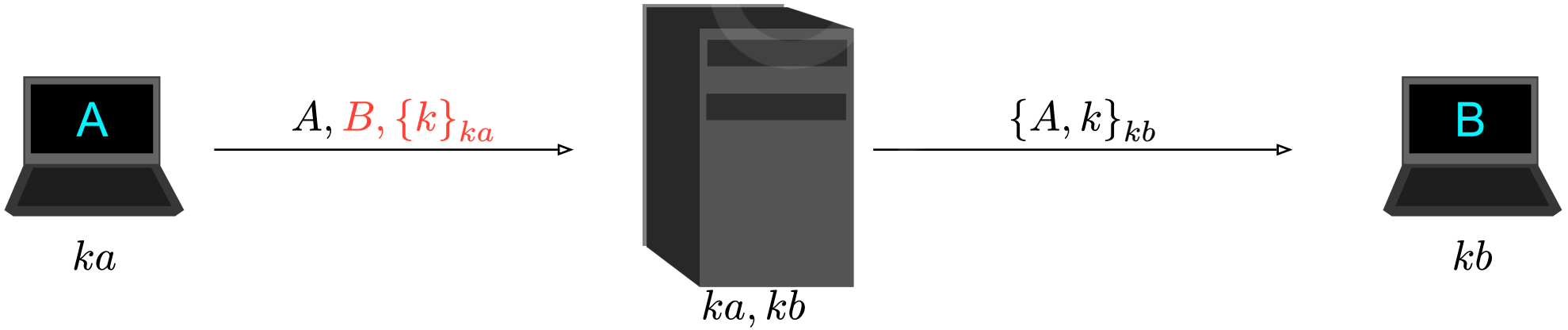
- Protocols implemented in Approxis;
  - proposed modelisation of protocols;
  - proved security for an example;
- implemented new cryptographic game: IND-CCA, INT-PTXT;
- other examples of cryptographic proof: attack on a protocol, KEMDEM, formalized abstract encryption schemes, security of One-Time Pad.

⇨ Pull request to clutch repository

Weakened WMF



Weakened WMF



The attack modelled in Approxis

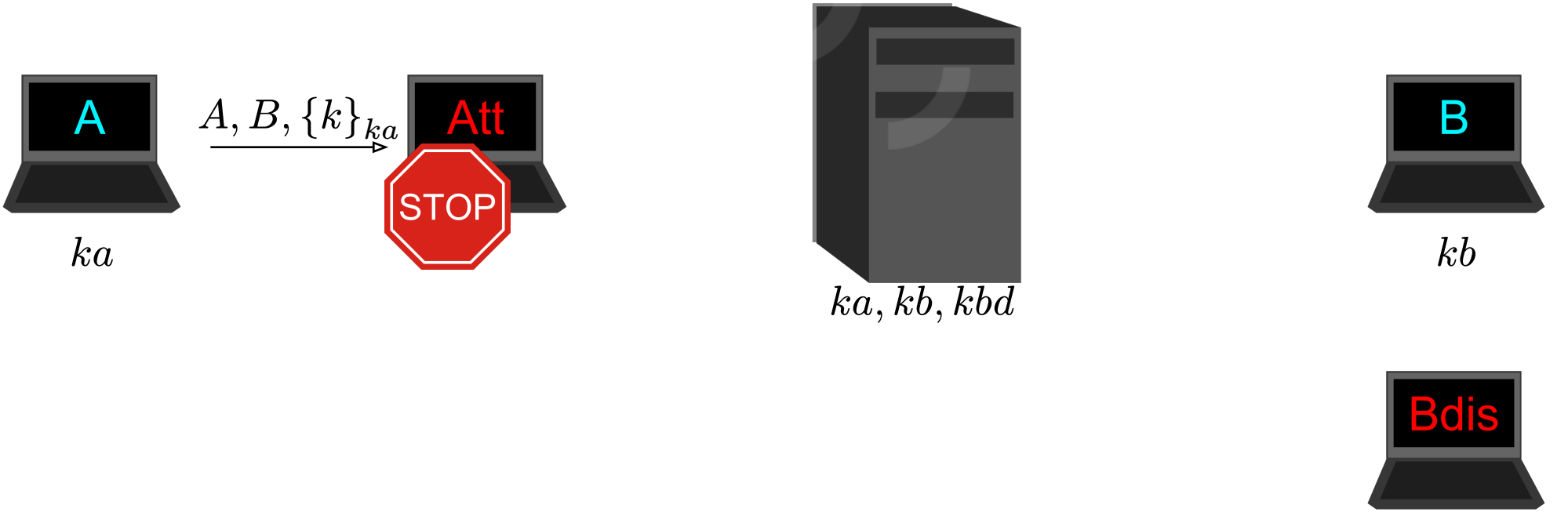
ka

$A, B, \{k\}_{ka}$

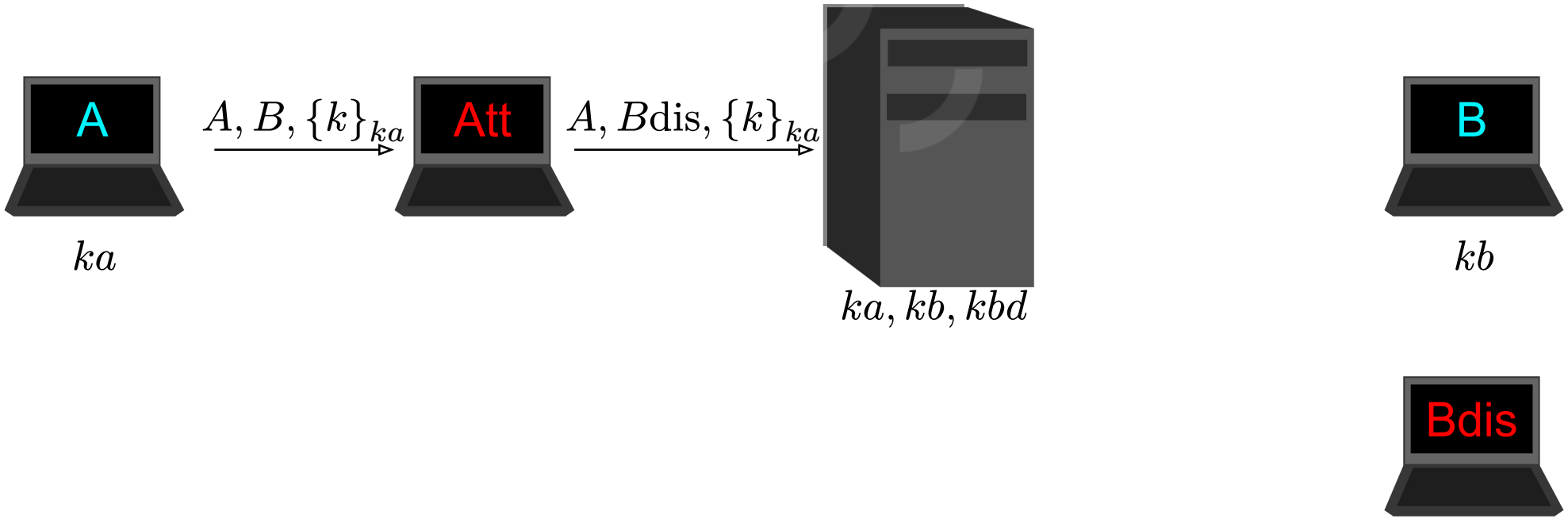
ka, kb

kb

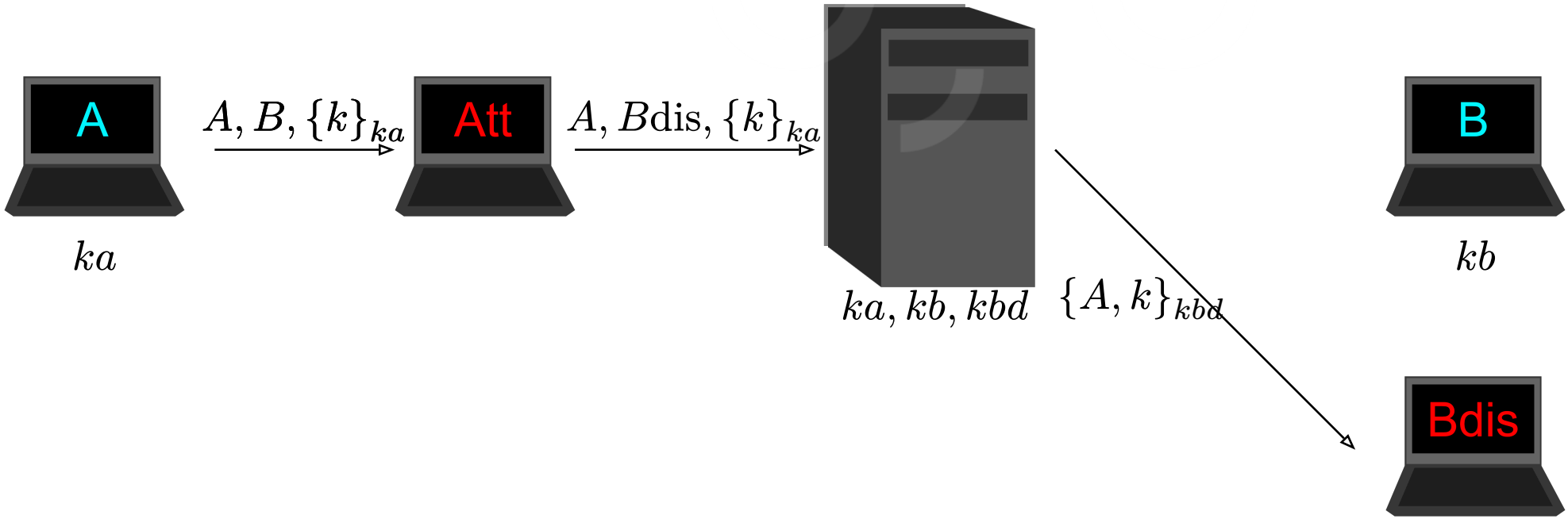
The attack modelled in Approxis



The attack modelled in Approxis



The attack modelled in Approxis



Separation logic bis

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Separation logic bis

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Main ingredients:

- Logical formulas e.g. propositional, first-order;

Separation logic bis

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Main ingredients:

- Logical formulas e.g. propositional, first-order;
- points-to predicate $l \mapsto 42$;

Separation logic bis

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Main ingredients:

- Logical formulas e.g. propositional, first-order;
- points-to predicate $l \mapsto 42$;
- Hoare triple $\{l \mapsto x\} l := 42 \{l \mapsto 42\}$;

Separation logic bis

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Main ingredients:

- Logical formulas e.g. propositional, first-order;
- points-to predicate $l \mapsto 42$;
- Hoare triple $\{l \mapsto x\} l := 42 \{l \mapsto 42\}$;
- separating conjunction

Separation logic bis

Goal: Reason about program with a heap (e.g. C or OCaml with references)

Main ingredients:

- Logical formulas e.g. propositional, first-order;
- points-to predicate $l \mapsto 42$;
- Hoare triple $\{l \mapsto x\} l := 42 \{l \mapsto 42\}$;
- separating conjunction: $l \mapsto 42 * l \mapsto 42$ must lead to a contradiction.

KEMDEM

Asymmetric/Public-key encryption: $\mathrm{aenc}, \mathrm{adec}, (\mathrm{pk}, \mathrm{sk}) \leftrightsquigarrow \mathrm{keygen}$

$\mathrm{adec}(\mathrm{sk}, \mathrm{aenc}(\mathrm{pk}, m)) = m$

KEMDEM

Asymmetric/Public-key encryption: $\mathrm{aenc}, \mathrm{adec}, (\mathrm{pk}, \mathrm{sk}) \leftrightsquigarrow \mathrm{keygen}$

$\mathrm{adec}(\mathrm{sk}, \mathrm{aenc}(\mathrm{pk}, m)) = m$

$\mathrm{senc}, \mathrm{sdec}, \mathrm{keygen}_s, \mathrm{aenc}, \mathrm{adec}, \mathrm{keygen}_a$:

$\mathrm{keygen} = \mathrm{keygen}_a$

KEMDEM

Asymmetric/Public-key encryption: $\mathrm{aenc}, \mathrm{adec}, (\mathrm{pk}, \mathrm{sk}) \leftrightsquigarrow \mathrm{keygen}$

$\mathrm{adec}(\mathrm{sk}, \mathrm{aenc}(\mathrm{pk}, m)) = m$

$\mathrm{senc}, \mathrm{sdec}, \mathrm{keygen}_s, \mathrm{aenc}, \mathrm{adec}, \mathrm{keygen}_a$:

$\mathrm{keygen} = \mathrm{keygen}_a$

$\mathrm{aenc2}(\mathrm{pk}, m) = (\mathrm{senc}(k, m), \mathrm{aenc}(\mathrm{pk}, k))$

KEMDEM

Asymmetric/Public-key encryption: $\mathsf{aenc}, \mathsf{adec}, (\mathsf{pk}, \mathsf{sk}) \leftrightsquigarrow \mathsf{keygen}$

$\mathsf{adec}(\mathsf{sk}, \mathsf{aenc}(\mathsf{pk}, m)) = m$

$\mathsf{senc}, \mathsf{sdec}, \mathsf{keygen}_s, \mathsf{aenc}, \mathsf{adec}, \mathsf{keygen}_a$:

$\mathrm{keygen} = \mathsf{keygen}_a$

$\mathsf{aenc2}(\mathsf{pk}, m) = (\mathsf{senc}(k, m), \mathsf{aenc}(\mathsf{pk}, k))$

$\mathsf{adec2}(\mathsf{sk}, m) = \text{let } \mathrm{encmsg}, \mathrm{enckey} = m \text{ in } \mathsf{sdec}(\mathsf{adec}(\mathsf{sk}, \mathrm{enckey}), \mathrm{encmsg})$