Coq workshop 2021

SSProve

Modular Cryptographic Proofs in Coq

Carmine **Abate** Philipp G. **Haselwarter** Exequiel **Rivas**

Antoine **Van Muylder** Théo **Winterhalter**

Cătălin **Hriţcu** Kenji **Maillard** Bas **Spitters**

2004

“Crisis of rigour” in crypto

Shoup, Bellare, Rogaway

Game playing proofs

2004

“Crisis of rigour” in crypto

Shoup, Bellare, Rogaway

Game playing proofs

A lot of good things

and more...

CertiCrypt

FCF

A lot of good things

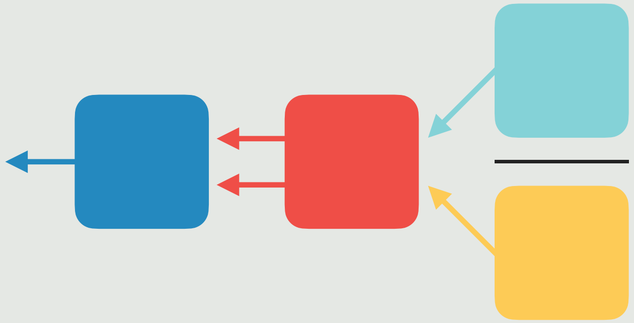
and more...

2018

State separating proofs SSP

Brzuska, Delignat-Lavaud, Fournet, Kohbrok, Kohlweiss

Modular/scalable high-level game playing proofs



2018

State separating proofs SSP

Brzuska, Delignat-Lavaud, Fournet, Kohbrok, Kohlweiss

Modular/scalable high-level game playing proofs

2021

SSProve

CSF 2021

Semantics for SSP + relational program logic

SSP + 

2021

SSProve

CSF 2021

Semantics for SSP + relational program logic

SSP methodology

SSP methodology

Real

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

SSP methodology

Real

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

Ideal

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  rand_msg ← sample ℳ
  (r, c) ← enc(key, rand_msg)
  return (r, c)
```

SSP methodology

Real

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

Ideal

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  rand_msg ← sample ℳ
  (r, c) ← enc(key, rand_msg)
  return (r, c)
```

SSP methodology

Real

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

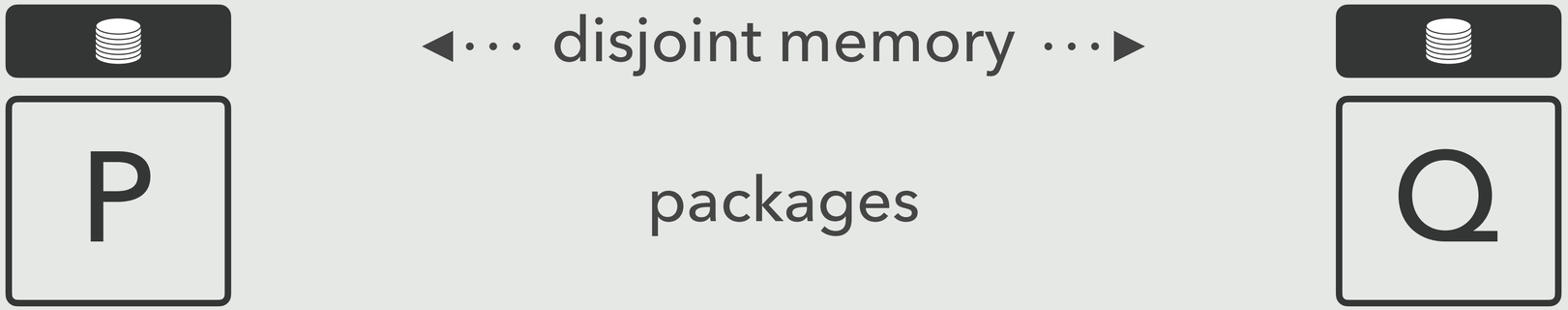
$\approx_{\varepsilon}$

Ideal

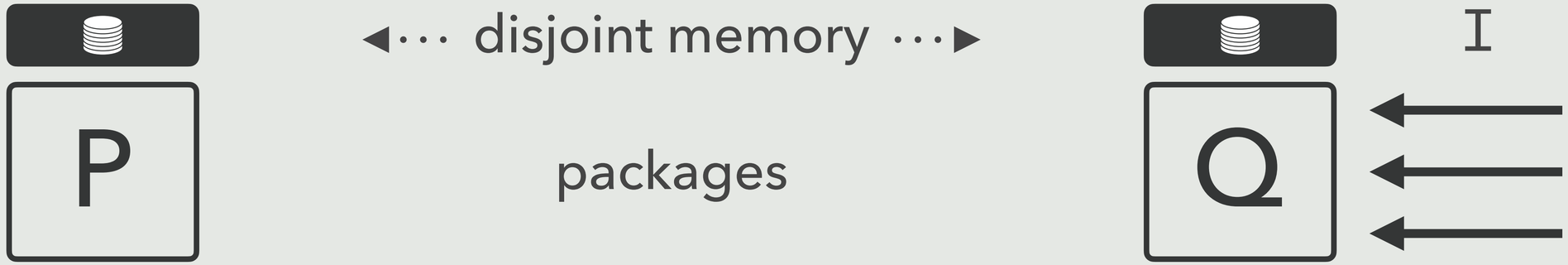
```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  rand_msg ← sample ℳ
  (r, c) ← enc(key, rand_msg)
  return (r, c)
```

SSP methodology

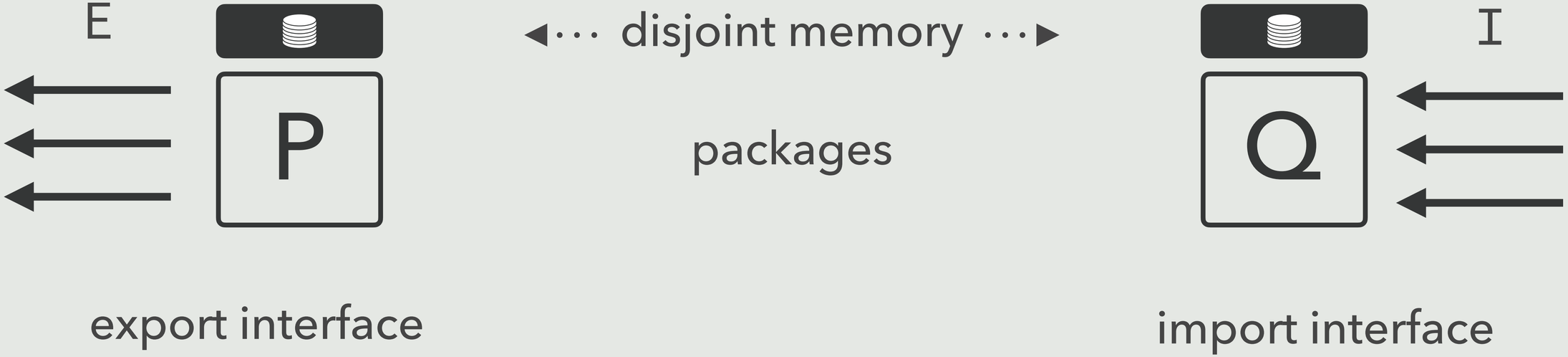
SSP methodology



SSP methodology

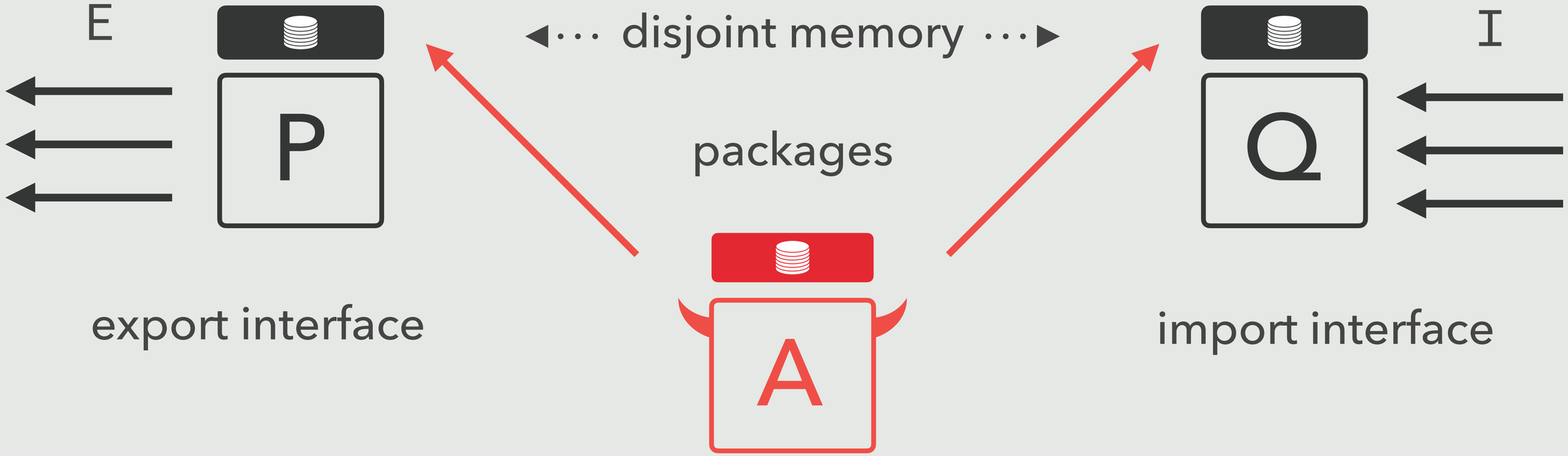


SSP methodology

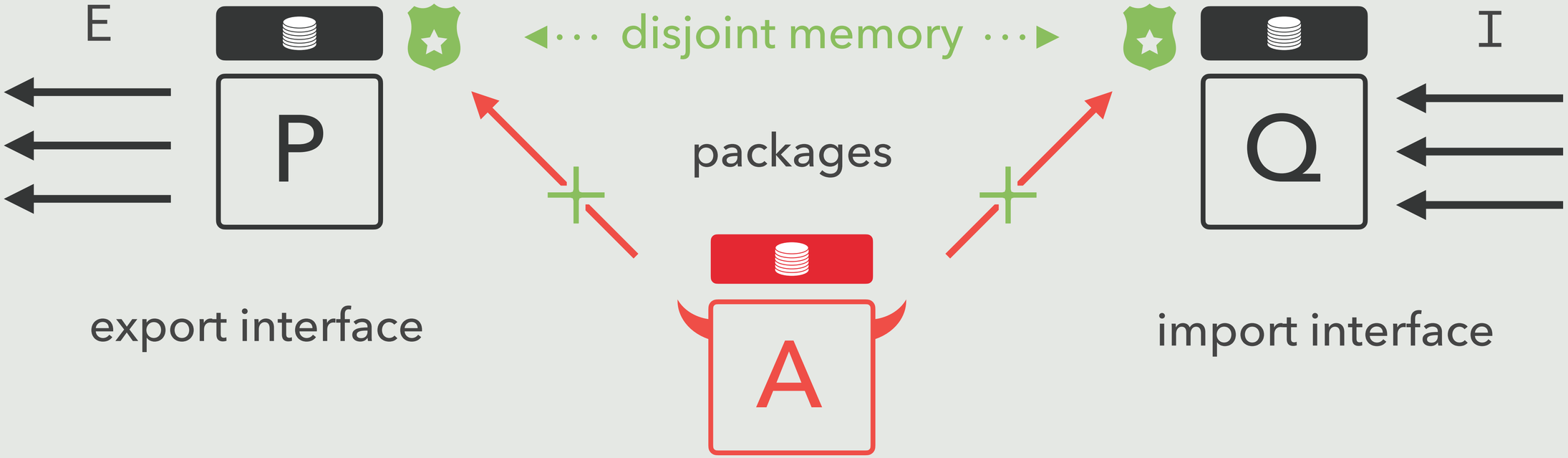


SSP methodology

SSP methodology



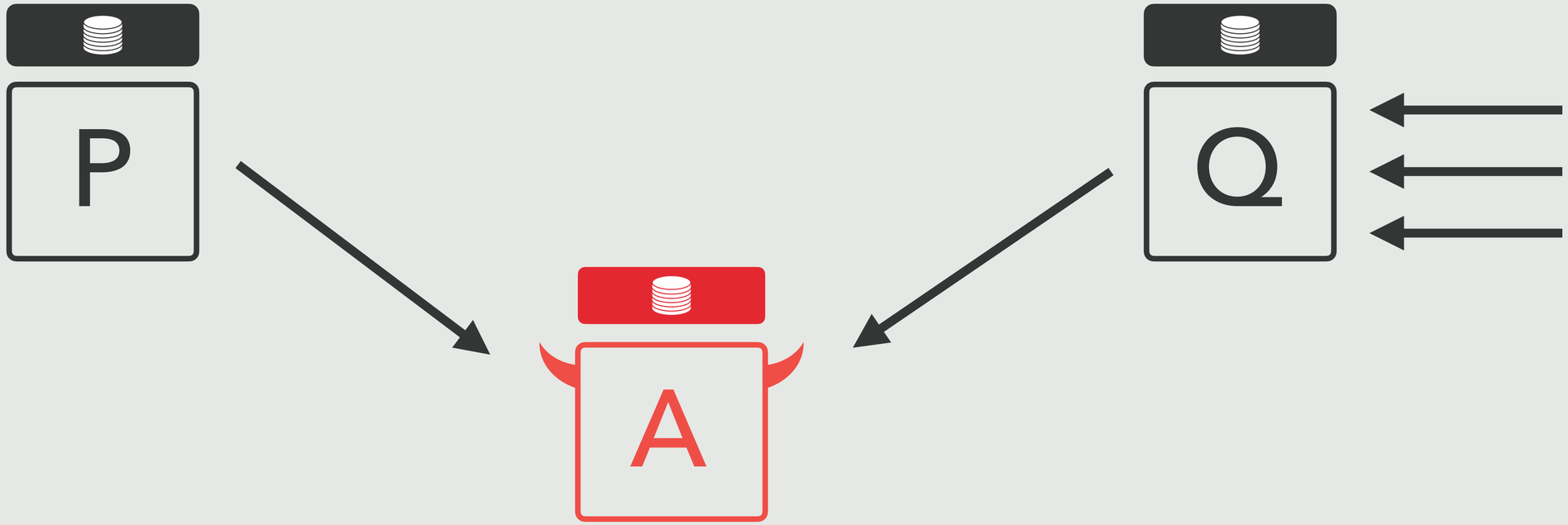
SSP methodology



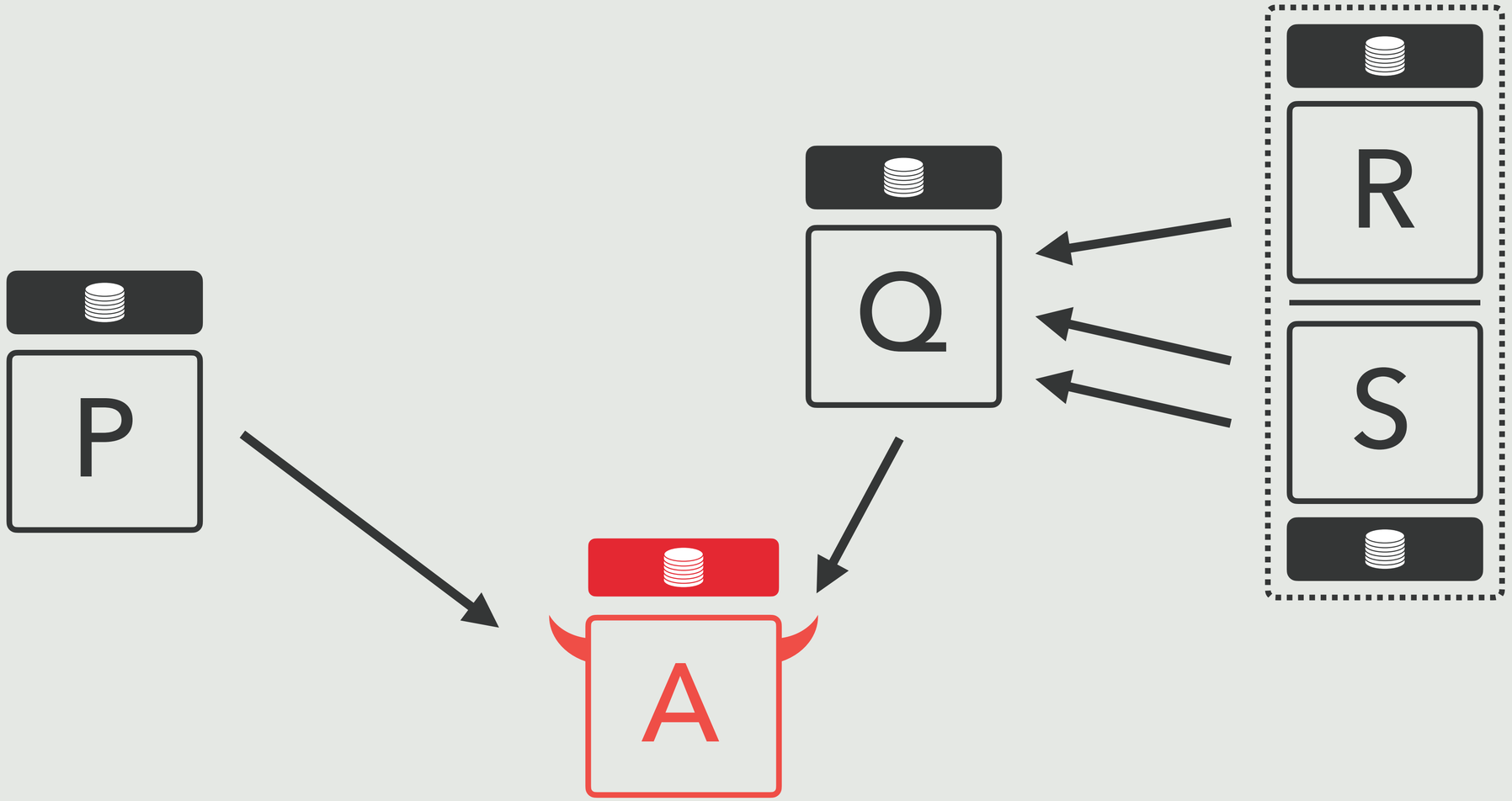
SSP methodology

SSP methodology

SSP methodology

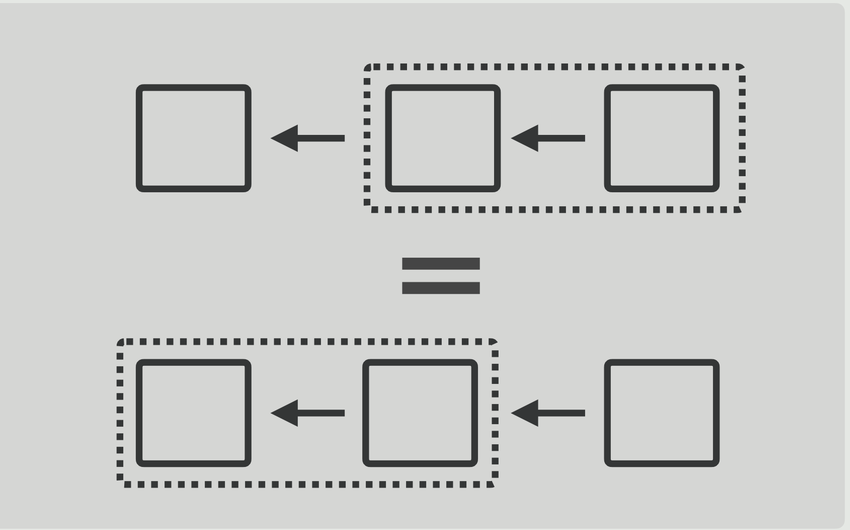


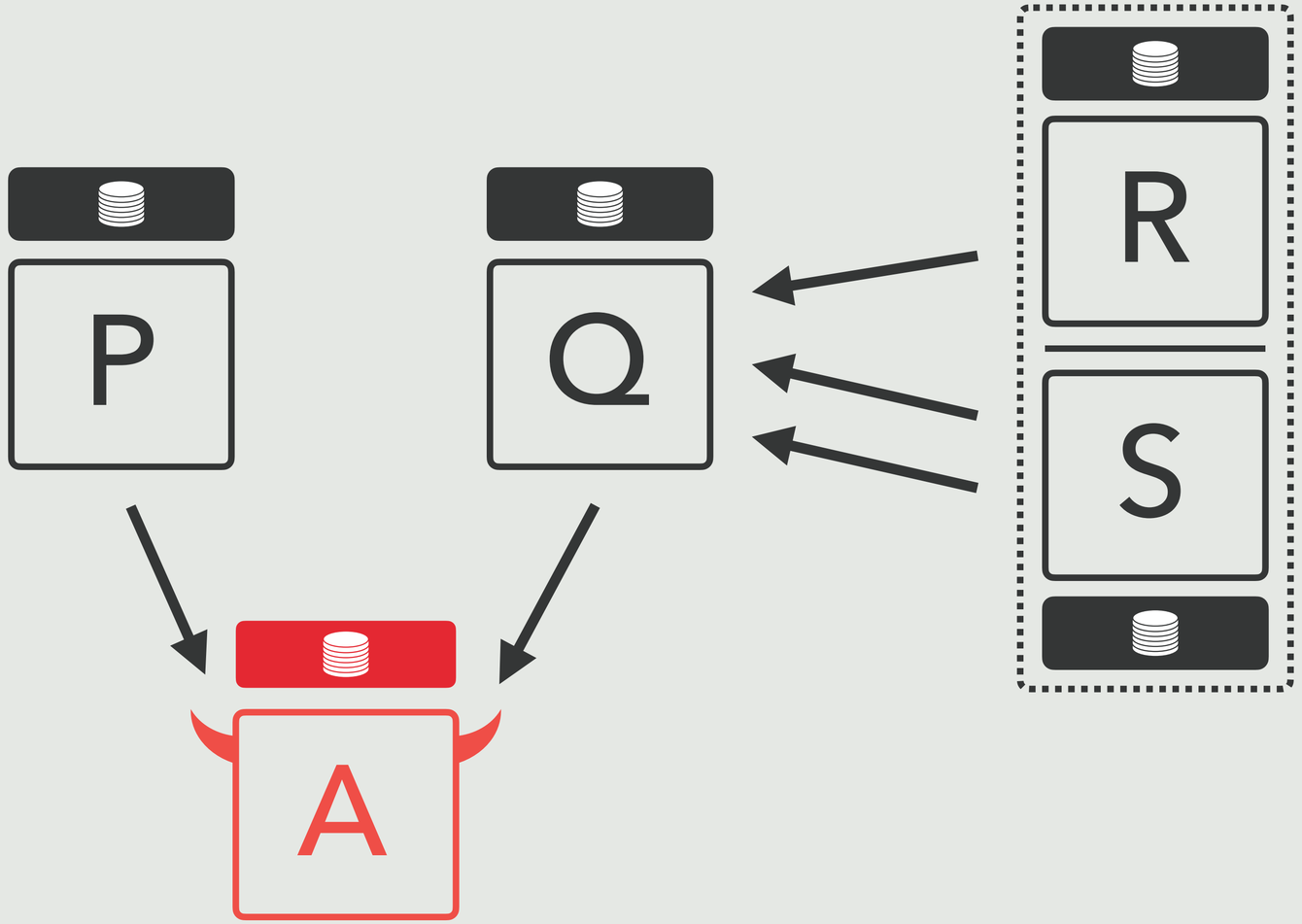
SSP methodology

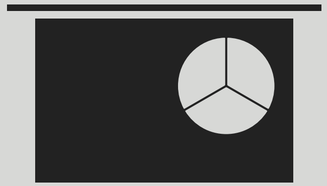


SSP methodology

package laws







More details

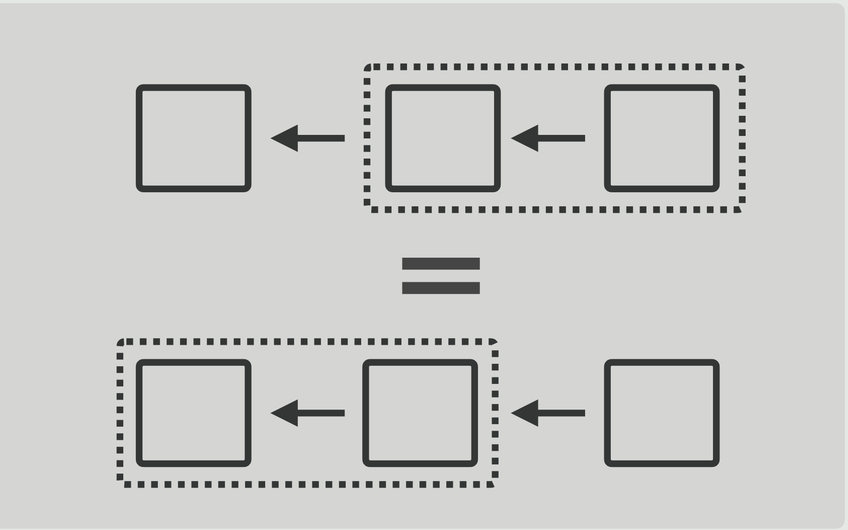
Philipp's talk at CSF 2021

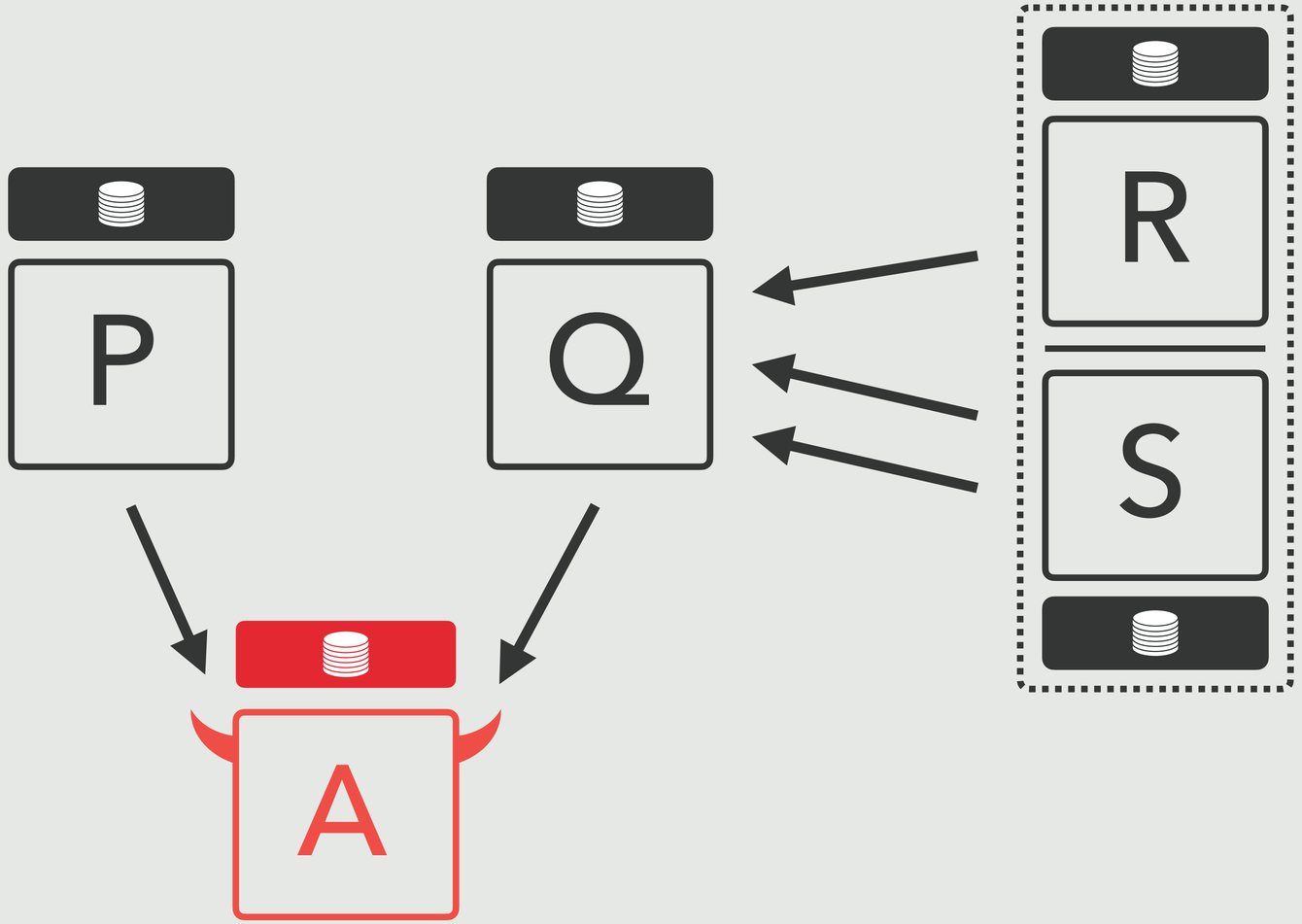
+

SSP paper

SSProve paper

package laws

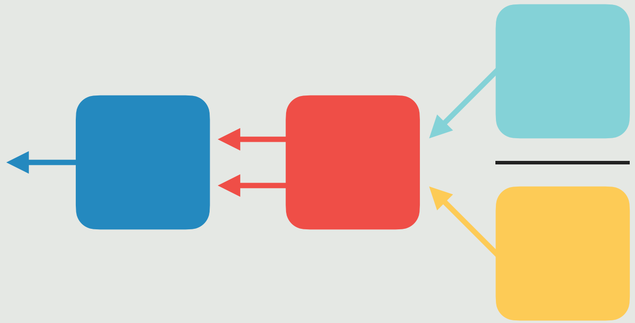




SSProve

formally defines

packages



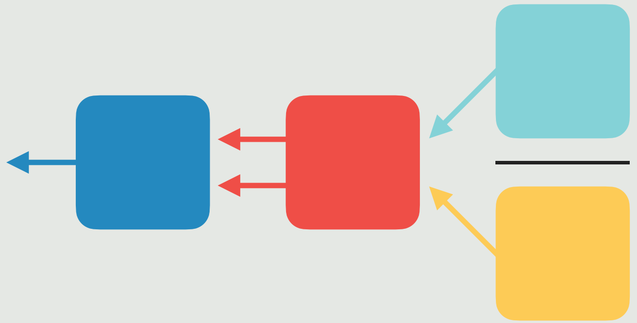
SSProve

formally defines

programs

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

packages



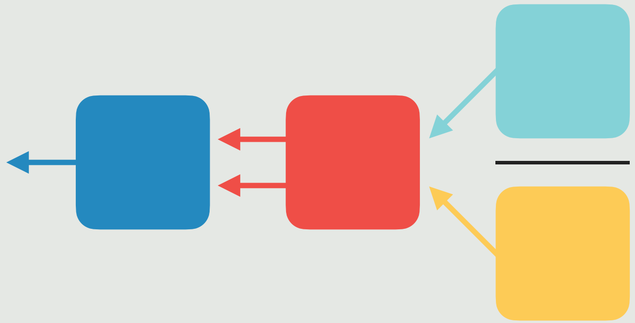
SSProve

formally defines

programs

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

packages



package laws

SSProve

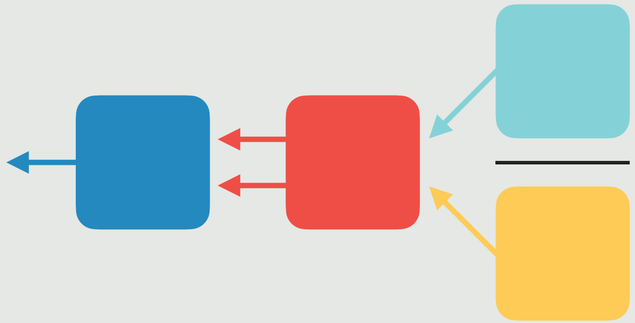
formally defines

programs

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

semantics

packages



package laws

SSProve

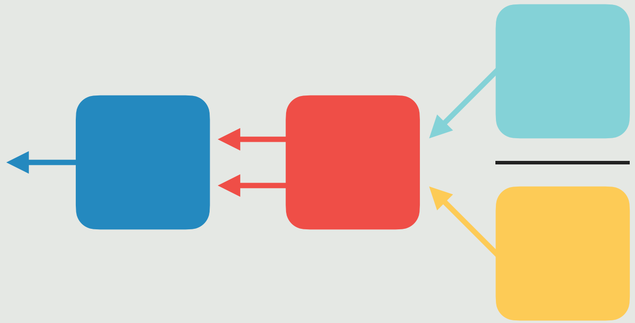
formally defines

programs

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

semantics

packages



package laws

security notion

SSProve

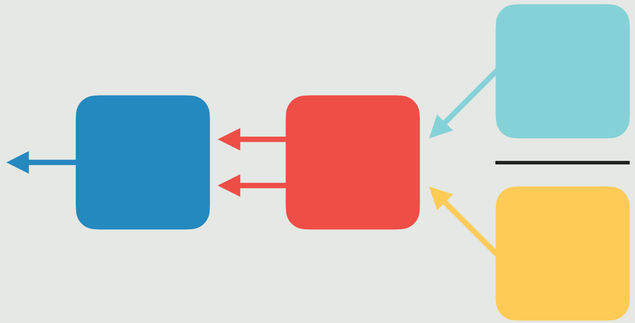
formally defines

programs

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

semantics

packages



package laws

security notion

```
Inductive raw_code A : Type :=
| ret (x : A)
```

```
Inductive raw_code A : Type :=
| ret (x : A)
| opr (o : opsig) (x : src o) (k : tgt o → raw_code A)
```

```
z ← op o · x ;; k z
```

```
Inductive raw_code A : Type :=
| ret (x : A)
| opr (o : opsig) (x : src o) (k : tgt o → raw_code A)
| getr (ℓ : Location) (k : ℓ → raw_code A)
```

```
z ← op o · x ;; k z
```

```
z ← get ℓ ;; k z
```

```
Inductive raw_code A : Type :=
| ret (x : A)
| opr (o : opsig) (x : src o) (k : tgt o → raw_code A)
| getr (ℓ : Location) (k : ℓ → raw_code A)
| putr (ℓ : Location) (v : ℓ) (k : raw_code A)
```

```
z ← op o · x ;; k z
```

```
z ← get ℓ ;; k z
```

```
put ℓ := v ;; k
```

```
Inductive raw_code A : Type :=
| ret (x : A)
| opr (o : opsig) (x : src o) (k : tgt o → raw_code A)
| getr (ℓ : Location) (k : ℓ → raw_code A)
| putr (ℓ : Location) (v : ℓ) (k : raw_code A)
| sampler (𝒟 : Op) (k : Arit 𝒟 → raw_code A).
```

```
z ← op o · x ;; k z
```

```
z ← get ℓ ;; k z
```

```
put ℓ := v ;; k
```

```
z ← sample 𝒟 ;; k z
```

```
Inductive raw_code A : Type :=
| ret (x : A)
| opr (o : opsig) (x : src o) (k : tgt o → raw_code A)
| getr (ℓ : Location) (k : ℓ → raw_code A)
| putr (ℓ : Location) (v : ℓ) (k : raw_code A)
| sampler (𝒟 : Op) (k : Arit 𝒟 → raw_code A).
```

```
#import {sig #[id] : 'nat → 'bool } as f ;; k f
```

```
z ← get ℓ ;; k z
```

```
put ℓ := v ;; k
```

```
z ← sample 𝒟 ;; k z
```

```
Inductive raw_code A : Type :=
| ret (x : A)
| opr (o : opsig) (x : src o) (k : tgt o → raw_code A)
| getr (ℓ : Location) (k : ℓ → raw_code A)
| putr (ℓ : Location) (v : ℓ) (k : raw_code A)
| sampler (𝒟 : Op) (k : Arit 𝒟 → raw_code A).
```

```
#import {sig #[id] : 'nat → 'bool } as f ;; k f
```

```
z ← get ℓ ;; k z
```

```
put ℓ := v ;; k
```

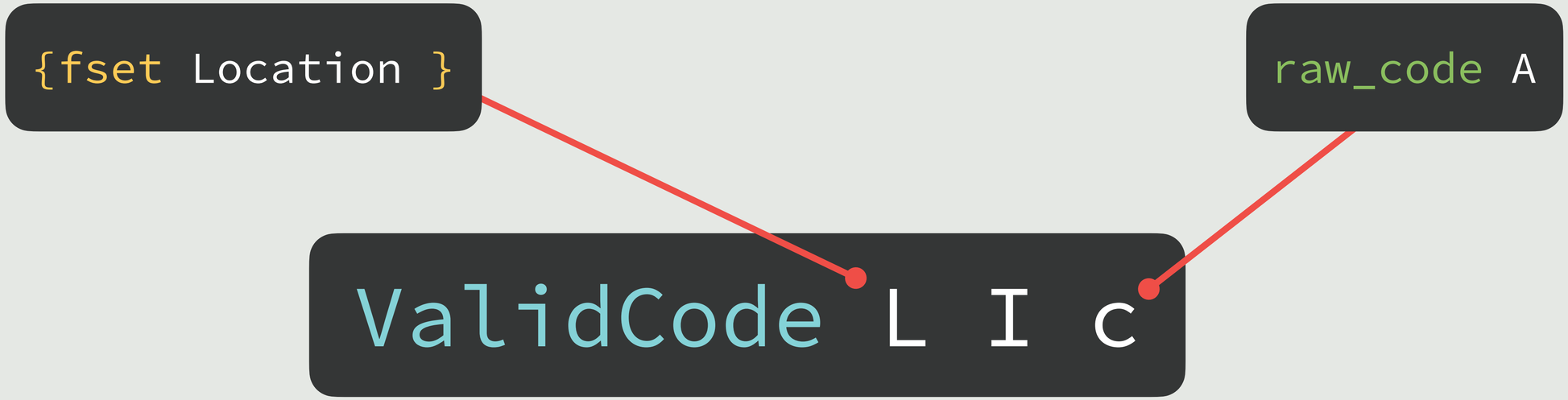
```
z ← sample 𝒟 ;; k z
```

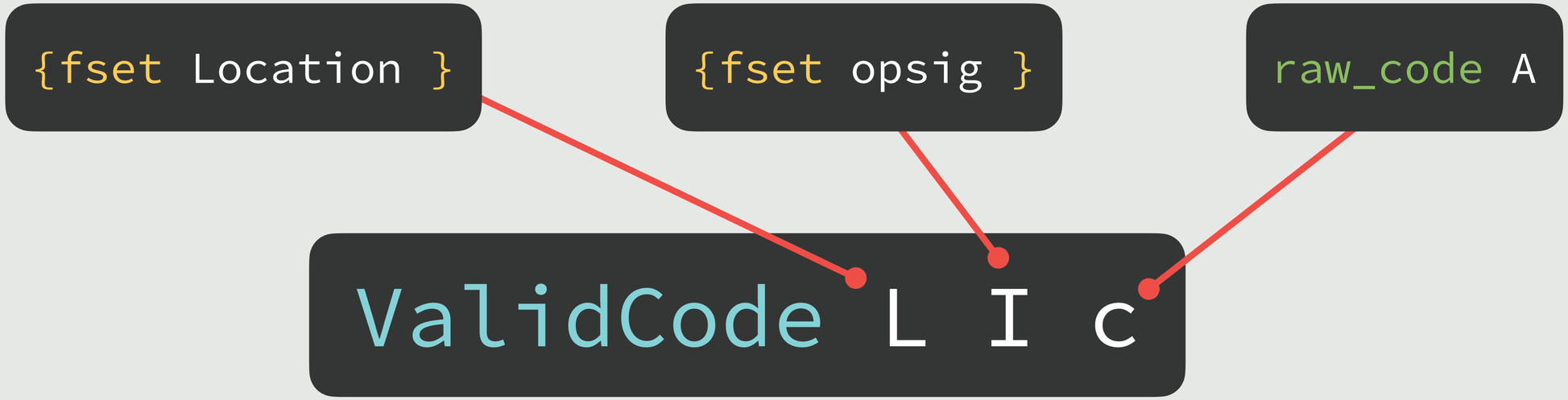
```
z ← m ;; k z
```

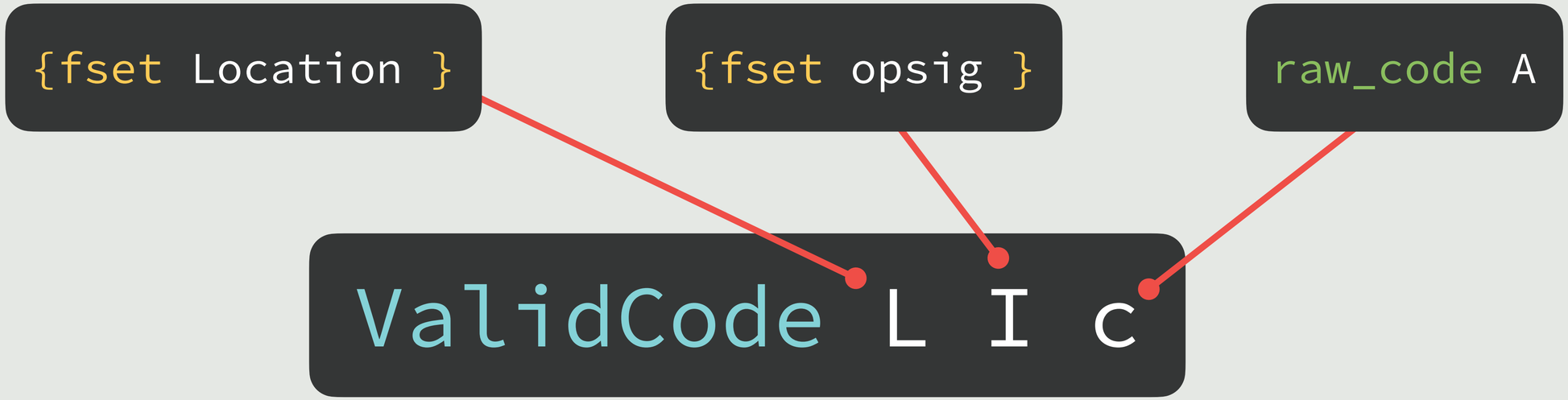
raw_code A

ValidCode L I c

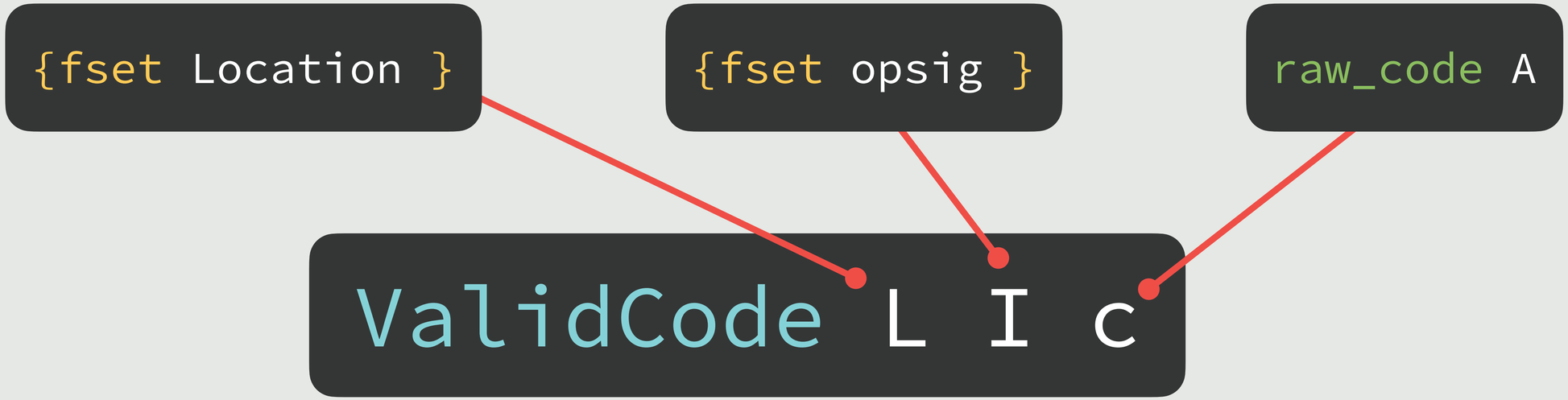

raw_code A
ValidCode L I c


{fset Location }
raw_code A
ValidCode L I c


{fset Location }
{fset opsig }
raw_code A
ValidCode L I c

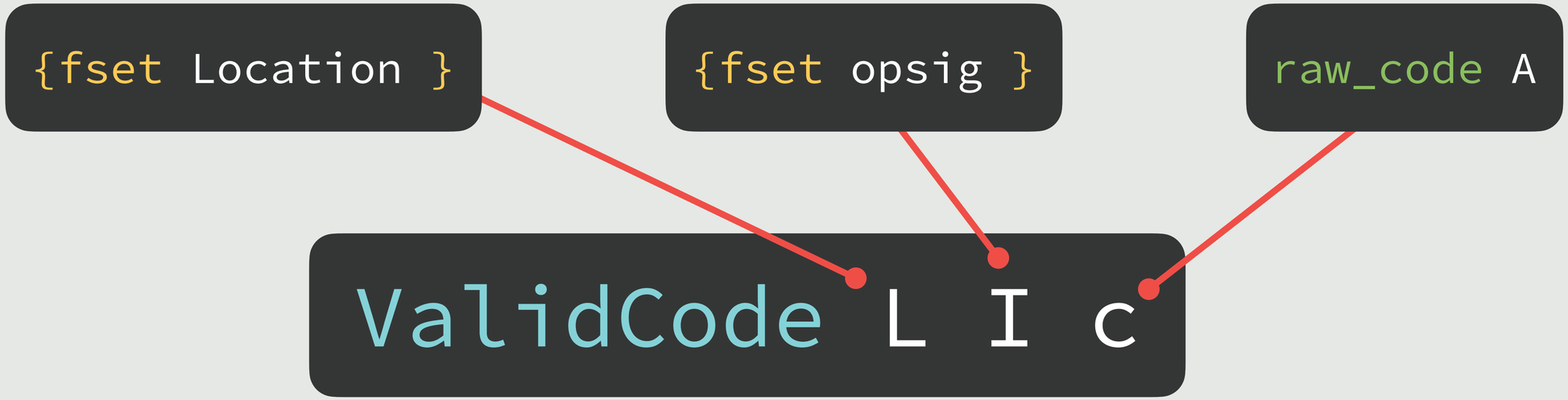



```
valid_ret : ∀ x, ValidCode L I (ret x)
```



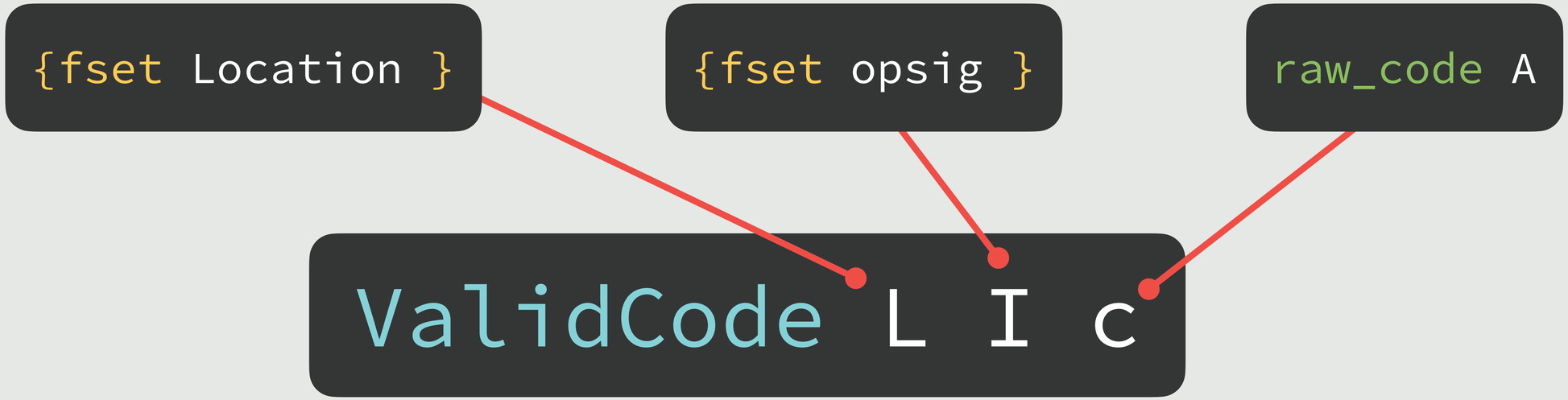

```
valid_ret : ∀ x, ValidCode L I (ret x)
```

```
valid_getr :
  ∀ ℓ k,
    ℓ \in L →
    (∀ v, ValidCode L I (k v)) →
    ValidCode L I (x ← get ℓ ;; k x)
```

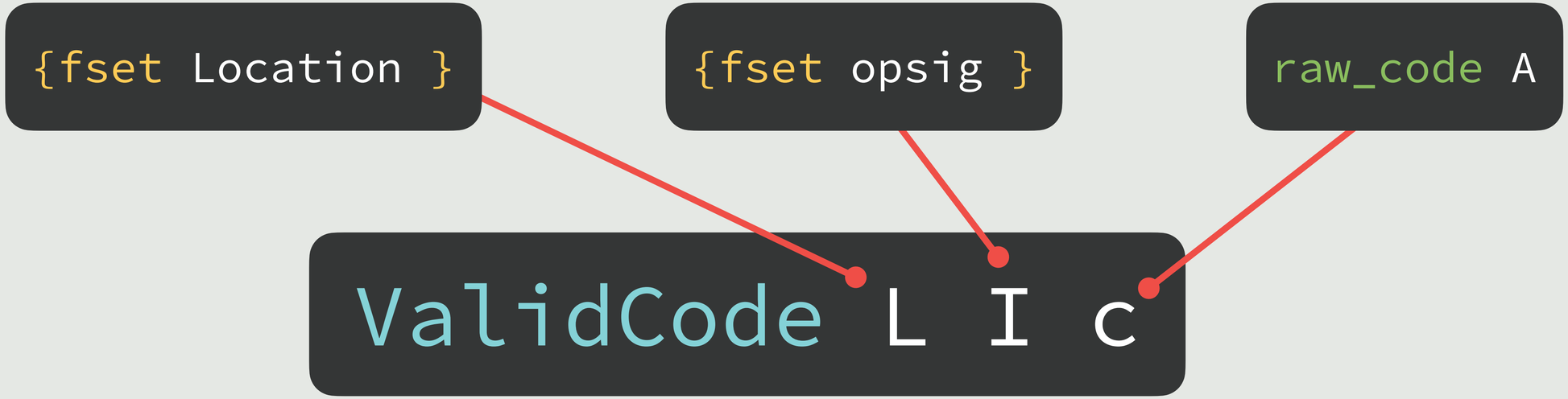



```
valid_ret : ∀ x, ValidCode L I (ret x)
```

```
valid_opr :
  ∀ o x k,
    o \in I →
    (∀ v, ValidCode L I (k v)) →
    ValidCode L I (z ← op o · x ;; k z)
```




```
ValidCode L I (if b then c₀ else c₁)
```




```
ValidCode L I (if b then c₀ else c₁)
```

```
#[export] Hint Extern 2 (ValidCode ?L ?I (match ?t with _ => _ end)) =>
  destruct t
  : typeclass_instances ssprove_valid_db.
```

```
Definition typed_raw_function :=
  Σ (S T : chUniverse), S → raw_code T.

Definition raw_package :=
  {fmap ident -> typed_raw_function }.
```

```
Definition typed_raw_function :=
  Σ (S T : chUniverse), S → raw_code T.

Definition raw_package :=
  {fmap ident -> typed_raw_function }.
```

```
Class ValidPackage L I E p :=
  ∀ o, o \in E →
    let '(id, (src, tgt)) := o in
    ∃ (f : src → raw_code tgt),
      p id = Some (src ; tgt ; f) ∧
      ∀ x, ValidCode L I (f x).
```

```
Definition typed_raw_function :=
  Σ (S T : chUniverse), S → raw_code T.

Definition raw_package :=
  {fmap ident -> typed_raw_function }.
```

```
Class ValidPackage L I E p :=
  ∀ o, o \in E →
    let '(id, (src, tgt)) := o in
    ∃ (f : src → raw_code tgt),
      p id = Some (src ; tgt ; f) ∧
      ∀ x, ValidCode L I (f x).
```

```
Record package L I E := mkpackage {
  pack : raw_package ;
  pack_valid : ValidPackage L I E pack
}.
```

```
Definition IND_CPA⁰ :
  package [fset key_location ] [interface]
    [interface val #[ ENC ] : 'word → 'word × 'word ] :=
  [package
    def #[ ENC ] (m : 'word) : 'word × 'word
    {
      k ← get key_location ;;
      match k with
      | None =>
        k_val ← sample uniform i_key ;;
        put key_location := Some k_val ;;
        enc m k_val
      | Some k_val =>
        enc m k_val
      end
    }
  ].
```

```
Definition IND_CPA⁰ :
  package [fset key_location ] [interface]
    [interface val #[ ENC ] : 'word → 'word × 'word ] :=
  [package
    def #[ ENC ] (m : 'word) : 'word × 'word
    {
      k ← get key_location ;;
      match k with
      | None =>
        k_val ← sample uniform i_key ;;
        put key_location := Some k_val ;;
        enc m k_val
      | Some k_val =>
        enc m k_val
      end
    }
  ].
```

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

Package equivalence

Real

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

$\approx_{\varepsilon}$

Ideal

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  rand_msg ← sample ℳ
  (r, c) ← enc(key, rand_msg)
  return (r, c)
```

Package equivalence

Real

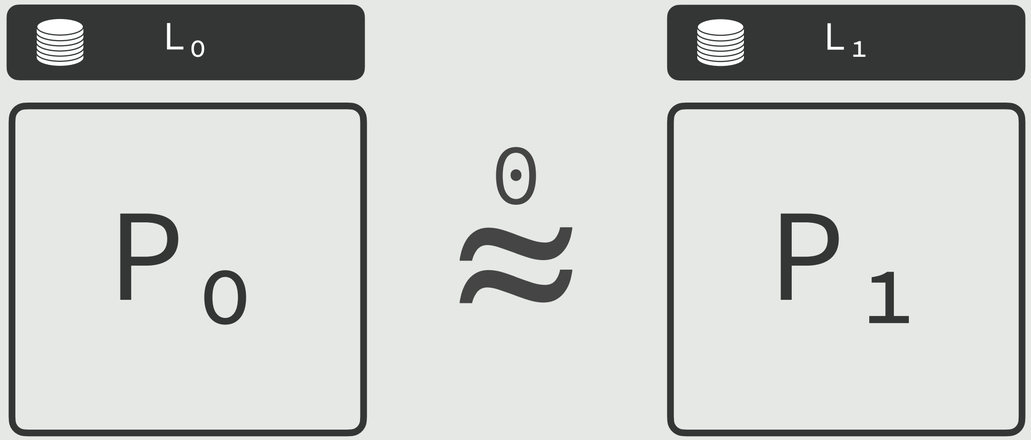
```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  (r, c) ← enc(key, msg)
  return (r, c)
```

$\approx_{\varepsilon}$

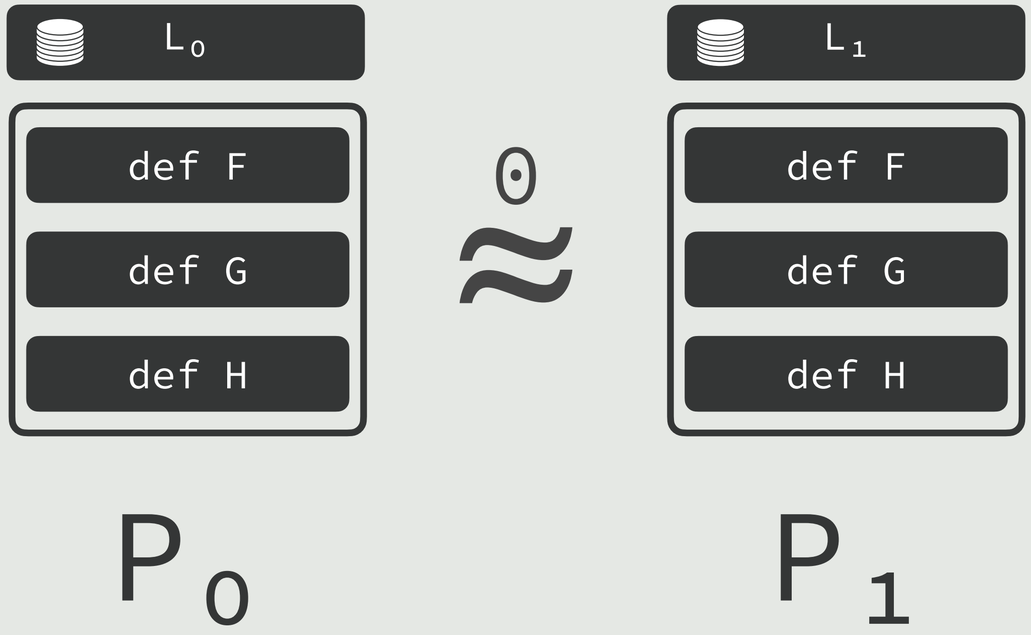
Ideal

```
ENC(msg) :=
  if key = None then
    key ← sample 𝒟
  rand_msg ← sample ℳ
  (r, c) ← enc(key, rand_msg)
  return (r, c)
```

Package equivalence



Package equivalence



If

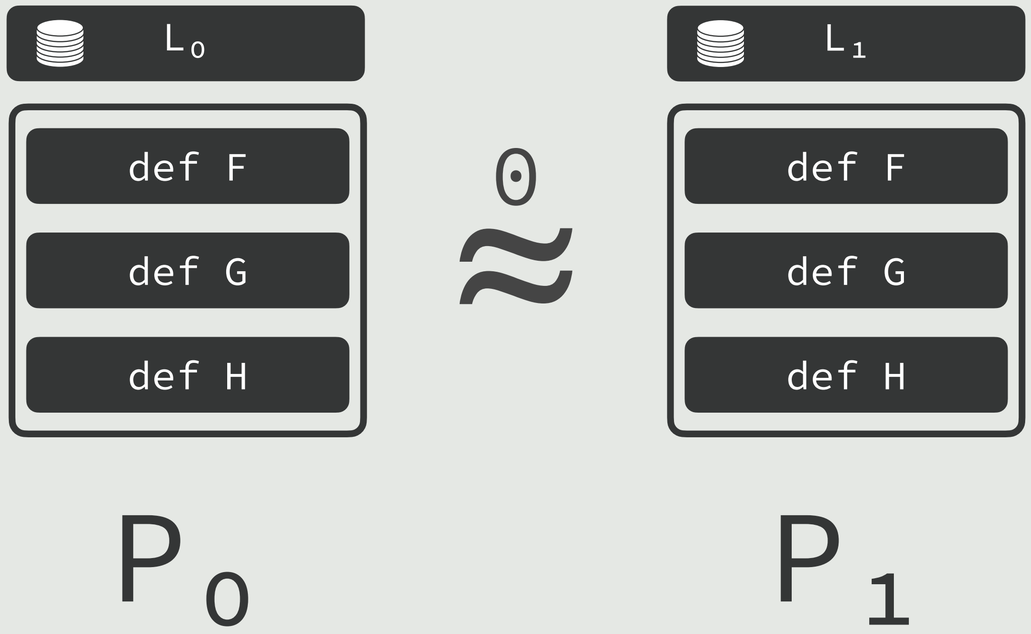
$$\forall\ x,\ \vdash P_0.F\ x \approx P_1.F\ x$$

$$\forall\ x,\ \vdash P_0.G\ x \approx P_1.G\ x$$

$$\forall\ x,\ \vdash P_0.H\ x \approx P_1.H\ x$$

modulo invariant ψ respecting/ignoring L_0, L_1

then



Probabilistic relational Hoare program logic

$$\forall\ x,\ \vdash\ P_0.F\ x \approx P_1.F\ x$$

modulo invariant ψ respecting/ignoring L_0, L_1

Probabilistic relational Hoare program logic

$$\forall\ x,\ \vdash\ \{\!|\ \mathtt{pre}\ |\!\}\ \mathtt{P}_0\mathtt{.F}\ x \approx \mathtt{P}_1\mathtt{.F}\ x\ \{\!|\ \mathtt{post}\ |\!\}$$

modulo invariant ψ respecting/ignoring $\mathtt{L}_0, \mathtt{L}_1$

Probabilistic relational Hoare program logic

$$\forall\ x,\ \vdash\ \{\!|\ \mathrm{pre}\ |\!\}\ P_0.F\ x \approx P_1.F\ x\ \{\!|\ \mathrm{post}\ |\!\}$$

$$\mathrm{pre} := \lambda\ '(s_0, s_1),\ \psi\ s_0\ s_1$$
$$\mathrm{post} := \lambda\ '(r_0, s_0)\ '(r_1, s_1),\ \psi\ s_0\ s_1 \wedge r_0 = r_1$$

ψ respecting/ignoring L_0, L_1

Probabilistic relational Hoare program logic

$$\forall\ x,\ \vdash\ \{\!|\ \mathit{pre}\ |\!\}\ P_0.F\ x \approx P_1.F\ x\ \{\!|\ \mathit{post}\ |\!\}$$

$$\begin{aligned}\mathit{pre}\ &:= \lambda\ '(s_0, s_1), \psi\ s_0\ s_1\\ \mathit{post}\ &:= \lambda\ '(r_0, s_0)\ '(r_1, s_1), \psi\ s_0\ s_1 \wedge r_0 = r_1\end{aligned}$$

ψ respecting/ignoring L_0, L_1

$$\psi\ \mathit{empty\_heap}\ \mathit{empty\_heap}$$

$$\begin{aligned}&\ell \setminus\!\mathrm{notin}\ L_0 \to \ell \setminus\!\mathrm{notin}\ L_1 \to \psi\ s_0\ s_1 \to\\ &\psi\ (\mathit{set\_heap}\ \ell\ v\ s_0)\ (\mathit{set\_heap}\ \ell\ v\ s_1)\end{aligned}$$

Probabilistic relational Hoare program logic

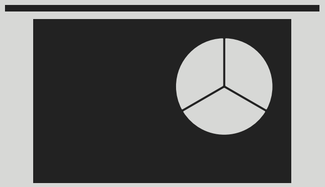
$$\vdash \{\!|\ \mathtt{pre}\ |\!\}\ \mathtt{c}_0 \approx \mathtt{c}_1\ \{\!|\ \mathtt{post}\ |\!\}$$

semantics with computation + specification monads

Probabilistic relational Hoare program logic

$$\vdash \{\!|\ \mathtt{pre}\ |\!\}\ \mathtt{c}_0 \approx \mathtt{c}_1\ \{\!|\ \mathtt{post}\ |\!\}$$

semantics with computation + specification monads



More details

Antoine’s talk at TYPES 2021

Examples

Synchronisation

Examples

Synchronisation

$$\vdash \{\!|\ \mathtt{pre}\ |\!\}\ x \leftarrow \mathtt{sample}\ D\ ;;\ k_0\ x \approx x \leftarrow \mathtt{sample}\ D\ ;;\ k_1\ x\ \{\!|\ \mathtt{post}\ |\!\}$$

Examples

Synchronisation

```
⊢ ⦃ pre ⦄ x ← sample D ;; k₀ x ≈ x ← sample D ;; k₁ x ⦃ post ⦄
```

```
ssprove_sync.
```

Examples

Synchronisation

```
⊢ ⦃ pre ⦄ x ← sample D ;; k₀ x ≈ x ← sample D ;; k₁ x ⦃ post ⦄
```

```
                                                    ssprove_sync.
```

```
∀ x, ⊢ ⦃ pre ⦄ k₀ x ≈ k₁ x ⦃ post ⦄
```

Examples

Synchronisation

Examples

Synchronisation

$$\vdash \{\!|\ \mathsf{pre}\ |\!\}\ x \leftarrow \mathsf{get}\ \ell\ ;;\ k_0\ x \approx x \leftarrow \mathsf{get}\ \ell\ ;;\ k_1\ x\ \{\!|\ \mathsf{post}\ |\!\}$$

Examples

Synchronisation

```
⊢ ⦃ pre ⦄ x ← get ℓ ;; k₀ x ≈ x ← get ℓ ;; k₁ x ⦃ post ⦄
```

```
ssprove_sync.
```

Examples

Synchronisation

```
⊢ ⦃ pre ⦄ x ← get ℓ ;; k₀ x ≈ x ← get ℓ ;; k₁ x ⦃ post ⦄
```

```
                                                    ssprove_sync.
```

```
∀ x, ⊢ ⦃ pre ⦄ k₀ x ≈ k₁ x ⦃ post ⦄
```

1.

```
get_pre_cond ℓ pre
```

2.

Examples

Synchronisation

```
⊢ ⦃ pre ⦄ x ← get ℓ ;; k₀ x ≈ x ← get ℓ ;; k₁ x ⦃ post ⦄
```

```
                                                    ssprove_sync.
```

```
∀ x, ⊢ ⦃ pre ⦄ k₀ x ≈ k₁ x ⦃ post ⦄
```

1.

```
get_pre_cond ℓ pre
```

2.

```
:= ∀ s₀ s₁, pre (s₀, s₁) → get_heap s₀ ℓ = get_heap s₁ ℓ
```

Examples

Synchronisation

```
⊢ ⦃ pre ⦄ x ← get ℓ ;; k₀ x ≈ x ← get ℓ ;; k₁ x ⦃ post ⦄
```

```
                                                    ssprove_sync.
```

```
∀ x, ⊢ ⦃ pre ⦄ k₀ x ≈ k₁ x ⦃ post ⦄
```

1.

```
get_pre_cond ℓ pre
```

2.

```
:= ∀ s₀ s₁, pre (s₀, s₁) → get_heap s₀ ℓ = get_heap s₁ ℓ
```

Examples

Swapping

```
⊢ ⦃ pre ⦄
  x ← get ℓ ;;
  y ← sample D ;;
  put ℓ := v ;;
  k x y
≈ c ⦃ post ⦄
```

Examples

Swapping

```
⊢ {| pre |}
0. x ← get ℓ ;;
1. y ← sample D ;;
2. put ℓ := v ;;
3. k x y
≈ c {| post |}
```

Examples

Swapping

```
⊢ ⦃ pre ⦄
0. x ← get ℓ ;;
1. y ← sample D ;;
2. put ℓ := v ;;
3. k x y
≈ c ⦃ post ⦄
```

```
ssprove_swap_lhs 1.
```

Examples

Swapping

```
⊢ ⦃ pre ⦄
0. x ← get ℓ ;;
2. put ℓ := v ;;
1. y ← sample D ;;
3. k x y
≈ c ⦃ post ⦄
```

```
ssprove_swap_lhs 1.
```

SSProve

SSP + $\vdash_{\approx}$ + 

Verified examples: PRF, ElGamal, KEM-DEM*

/SSProve/ssprove

links to videos in the README