SSProve: A Foundational Framework for Modular Cryptographic Proofs in Coq

Carmine Abate[1] **Philipp G. Haselwarter[2]** Exequiel Rivas[3] Antoine Van Muylder[4] Théo Winterhalter[1]
Cătălin Hrițcu[1] Kenji Maillard[5] Bas Spitters[2]

34th Computer Security Foundation Symposium,
24.06.2021

This work was in part supported by the European Research Council under ERC Starting Grant SECOMP (715753), by AFOSR grant *Homotopy type theory and probabilistic computation* (12595060), and by the Concordium Blockchain Research Center at Aarhus University. Antoine Van Muylder holds a PhD Fellowship from the Research Foundation – Flanders (FWO).

[1]MPI-SP [2]Aarhus University [3]Inria Paris [4]Vrije Universiteit Brussel [5]Inria Rennes

Why SSProve?

Motivation:

- Shoup, Bellare and Rogaway (2004): "Crisis of rigour" in cryptography.
  Proposal: Game-playing proofs
- Monolithic game-based proofs can become intractable
- *State-Separating Proofs* (SSP) from high-level structure of miTLS paper proofs (Brzuska, Delignat-Lavaud, Fournet, Kohbrok, Kohlweiss; 2018)

Why SSProve?

Motivation:

- Shoup, Bellare and Rogaway (2004): "Crisis of rigour" in cryptography.
  Proposal: Game-playing proofs
- Monolithic game-based proofs can become intractable
- *State-Separating Proofs* (SSP) from high-level structure of miTLS paper proofs
  (Brzuska, Delignat-Lavaud, Fournet, Kohbrok, Kohlweiss; 2018)

Contributions:

- Give precise meaning to SSP and formalise it in Coq prover
- Modular language, logic & semantics
- Theorem connecting high-level SSP arguments and low-level program logic
- Approach validated by formalising several examples
  This paper: PRF, ElGamal. Github: KEM-DEM, Σ-protocols (with N. Sidorenco).

Why SSProve?

Motivation:

- Shoup, Bellare and Rogaway (2004): "Crisis of rigour" in cryptography.
  Proposal: Game-playing proofs
- Monolithic game-based proofs can become intractable
- *State-Separating Proofs* (SSP) from high-level structure of miTLS paper proofs
  (Brzuska, Delignat-Lavaud, Fournet, Kohbrok, Kohlweiss; 2018)

Contributions:

- Give precise meaning to SSP and formalise it in Coq prover
- Modular language, logic & semantics
- Theorem connecting high-level SSP arguments and low-level program logic
- Approach validated by formalising several examples
  This paper: PRF, ElGamal. Github: KEM-DEM, Σ-protocols (with N. Sidorenco).

Requirements: IND-CPA security for PRF based encryption

To prove...

```
package:  IND-CPA^0
mem:  key :  option KEY

ENC(msg):
 if key = ⊥ then
   key <$ uniform {0,1}^n
 (r,c) ← enc(key, msg)
 return (r,c)
```

$\overset{\varepsilon}{\approx}$

```
package:  IND-CPA^1
mem:  key :  option KEY

ENC(msg):
 if key = ⊥ then
   key <$ uniform {0,1}^n
 msg_rnd <$ uniform {0,1}^n
 (r,c) ← enc(key, msg_rnd)
 return (r,c)
```

We need to...

Requirements: IND-CPA security for PRF based encryption

To prove...

```
package:  IND-CPA^0
mem:  key :  option KEY

ENC(msg):
 if key = ⊥ then
   key <$ uniform {0,1}^n
 (r,c) ← enc(key, msg)
 return (r,c)
```

$\overset{\varepsilon}{\approx}$

```
package:  IND-CPA^1
mem:  key :  option KEY

ENC(msg):
 if key = ⊥ then
   key <$ uniform {0,1}^n
 msg_rnd <$ uniform {0,1}^n
 (r,c) ← enc(key, msg_rnd)
 return (r,c)
```

We need to...

1. pick a proof assistant
2. define a core language (syntax, semantics)
3. prove code-level reasoning principles (pRHL)
4. define packages, package composition
5. define games, adversaries, and security
6. prove high-level reasoning principles (SSP)

Coq – a mature formal proof management system

Provides a formal language for

- mathematical definitions & theorems
- executable algorithms (pure, i.e. no state/probabilities etc)

Example libraries

- computer science: CompCert (C compiler), Verified Software Toolchain (verification of C programs), Fiat-Crypto (fast cryptographic primitives)
- mathematics: 4 colour theorem, Feit-Thompson theorem, real analysis

Architecture

- trusted code base = clearly delimited kernel
- tactic language for programming automation
- easy installation via package manager

SSProve/Core language

```
package:  IND-CPA⁰
mem:  key : option KEY
```

```
ENC(msg):
 if !key == ⊥ then
   k <$ uniform {0,1}^n
   key := k
 (r,c) ← enc(key, msg)
 return (r,c)
```

`!`ℓ	read from memory location ℓ
ℓ `:= v`	write `v` to memory location ℓ
`x <$ D`	sample from (sub-) distribution `D`
`x ← p(a)`	call imported procedure `p` on value `a`
c_1 `;` c_2	sequencing (omitted at end of line)
`return v`	embed `v` from Coq's ambient algorith. language

SSProve/Core language

```
package:  IND-CPA⁰
mem:  key : option KEY

ENC(msg):
 if !key == ⊥ then
   k <$ uniform {0,1}ⁿ
   key := k
 (r,c) ← enc(key, msg)
 return (r,c)
```

$!\ell$	read from memory location ℓ
ℓ := v	write v to memory location ℓ
x <$ D	sample from (sub-) distribution D
x ← p(a)	call imported procedure p on value a
c_1 ; c_2	sequencing (omitted at end of line)
return v	embed v from Coq's ambient algorith. language

Under the hood, in Coq:

$$\texttt{Inductive code A} = \textit{ret}\ (\texttt{x} : \texttt{A}) \mid \textit{call}\ (\texttt{p} : \texttt{op})\ (\texttt{x} : \texttt{src p})\ (\kappa : \texttt{tgt p} \rightarrow \texttt{code A}) \mid \ \ldots$$

SSProve/Core language

```
package:  IND-CPA^0
mem:  key :  option KEY

ENC(msg):
  if !key == ⊥ then
    k <$ uniform {0,1}^n
    key := k
  (r,c) ← enc(key, msg)
  return (r,c)
```

`!`ℓ	read from memory location ℓ
ℓ `:=` v	write v to memory location ℓ
x `<$` D	sample from (sub-) distribution D
x $\leftarrow$ p(a)	call imported procedure p on value a
c_1 ; c_2	sequencing (omitted at end of line)
`return` v	embed v from Coq's ambient algorith. language

Under the hood, in Coq:

$\texttt{Inductive code A} = \textit{ret}\ (\texttt{x} : \texttt{A}) \mid \textit{call}\ (\texttt{p} : \texttt{op})\ (\texttt{x} : \texttt{src p})\ (\kappa : \texttt{tgt p} \rightarrow \texttt{code A}) \mid \ldots$

Derived definitions:

```
assert(b):
  if b == false then
    BOOM <$ null distr {0,1}
    return BOOM
```

```
for(n, c):
  if n > 0 then
    c n
    for(n-1, c)
```

...

Rules of pRHL

Each rule is a theorem in Coq.

Details on semantics: Antoine Van Muylder's video presentation on SSProve at **TYPES 2021**.

$$\frac{c : \mathtt{code}\ L\ A}{\vDash \{m_0 = m_1\}\ c \sim c\ \{(r_0, r_1).\ m_0 = m_1 \land r_0 = r_1\}}\ \texttt{reflexivity}$$

$$\frac{\begin{array}{c} c_0 : \mathtt{code}\ L_0\ A_0 \quad c_1 : \mathtt{code}\ L_1\ A_1 \\ f_0 : A_0 \to\ \mathtt{code}\ L_0\ B_0 \quad f_1 : A_1 \to\ \mathtt{code}\ L_1\ B_1 \\ \vDash \{pre\}\ c_0 \sim c_1\ \{\mu\} \\ \forall a_0 a_1.\ \vDash \{(h_0, h_1).\ \mu(a_0, h_0)(a_1, h_1)\}\ (f_0\ a_0) \sim (f_1\ a_1)\ \{post\} \end{array}}{\vDash \{pre\}\ a_0 \leftarrow c_0;;\ f_0\ a_0 \ \sim\ a_1 \leftarrow c_1;;\ f_1\ a_1\ \{post\}}\ \texttt{seq}$$

$$\frac{\begin{array}{c} c_0 : \mathtt{code}\ L\ A_0 \quad c_1 : \mathtt{code}\ L\ A_1 \\ \vDash \{I\}\ c_0 \sim c_1\ \{(a_0, a_1).\ I \land post(a_0, a_1)\} \\ \vDash \{I\}\ c_1 \sim c_0\ \{(a_1, a_0).\ I \land post(a_0, a_1)\} \end{array}}{\vDash \{I\}\ c_0;;\ c_1 \sim c_1;;\ c_0\ \{(a_0, a_1).\ I \land post(a_0, a_1)\}}\ \texttt{swap}$$

$$\frac{\begin{array}{c} c_0\ c_0' : \mathtt{code}\ L\ A_0 \quad c_1 : \mathtt{code}\ J\ A_1 \\ \vDash \{pre\}\ c_0 \sim c_1\ \{post\} \qquad \forall h.\ \theta(c_0\ h) = \theta(c_0'\ h) \end{array}}{\vDash \{pre\}\ c_0' \sim c_1\ \{post\}}\ \texttt{eqDistrL}$$

$$\frac{\begin{array}{c} c_0 : \mathtt{code}\ L\ A_0 \quad c_1 : \mathtt{code}\ L\ A_1 \\ \vDash \{pre\}\ c_0 \sim c_1\ \{post\} \end{array}}{\vDash \{pre^{-1}\}\ c_1 \sim c_0\ \{post^{-1}\}}\ \texttt{symmetry}$$

$$\frac{\begin{array}{c} c_0, c_1 : \mathbb{N} \to \mathtt{code}\ L\ \mathtt{unit} \quad N : \mathbb{N} \\ \forall n. \vDash \{I\ n\}\ c_0 \sim c_1\ \{I\ (n+1)\} \end{array}}{\vDash \{I\ 0\}\ \mathtt{for\_loop}\ N\ c_0 \sim \mathtt{for\_loop}\ N\ c_1\ \{I\ (N+1)\}}\ \texttt{for-loop}$$

$$\frac{\begin{array}{c} c_0, c_1 : \mathtt{code}\ L\ \mathtt{bool} \quad N : \mathbb{N} \\ \vDash \{I(\mathtt{true}, \mathtt{true})\}\ c_0 \sim c_1\ \{(b_0, b_1).\ b_0 = b_1 \land I(b_0, b_1)\} \end{array}}{\begin{array}{c} \vDash \{I(\mathtt{true}, \mathtt{true})\}\ \mathtt{do\_while}\ N\ c_0 \sim \\ \mathtt{do\_while}\ N\ c_1\ \{(b_0, b_1).\ b_0 = b_1 = \mathtt{false} \lor I(\mathtt{false}, \mathtt{false})\} \end{array}}\ \texttt{do-while}$$

$$\frac{\begin{array}{c} c_0 : \mathtt{code}\ L_0\ A_0 \quad c_1 : \mathtt{code}\ L_1\ A_1 \\ \forall (h_0, h_1),\ pre_s(h_0, h_1) \Rightarrow pre_w(h_0, h_1), \quad \vDash \{pre_w\}\ c_0 \sim c_1\ \{post\} \end{array}}{\vDash \{pre_s\}\ c_0 \sim c_1\ \{post\}}\ \texttt{pre\_rule}$$

$$\frac{\begin{array}{c} c_0 : \mathtt{code}\ L_0\ A_0 \quad c_1 : \mathtt{code}\ L_1\ A_1 \\ \vDash \{pre\}\ c_0 \sim c_1\ \{post_s\} \\ \forall (a_0, h_0)(a_1, h_1),\ post_s(a_0, h_0)(a_1, h_1) \Rightarrow post_w(a_0, h_0)(a_1, h_1) \end{array}}{\vDash \{pre\}\ c_0 \sim c_1\ \{post_w\}}\ \texttt{post\_rule}$$

$$\frac{|A|, |B| < \omega \quad f : A \to B\ \mathit{bijective}}{\vDash \{pre\}\ a <\$\ \mathcal{U}(A) \sim b <\$\ \mathcal{U}(B)\ \{(a, b).\ f(a) = b \land pre\}}\ \texttt{uniform}$$

$$\frac{b_0, b_1 : \mathtt{bool}}{\vDash \{b_0 = b_1\}\ \mathtt{assert}\ \ b_0 \sim \mathtt{assert}\ \ b_1\ \{b_0 = \mathtt{true} \land b_1 = \mathtt{true}\}}\ \texttt{asrt}$$

$$\frac{b : \mathtt{bool}}{\vDash \{b = \mathtt{true}\}\ \mathtt{assert}\ \ b \sim \mathtt{return}\ \ ()\ \{b = \mathtt{true}\}}\ \texttt{asrtL}$$

$\vdots \qquad \vdots \qquad \vdots$

SSProve/Packages

```
package:  IND-CPA⁰
mem:  key :  option KEY
```

```
ENC(msg):
 if !key == ⊥ then
   k <$ uniform {0,1}^n
   key := k
 (r,c) ← enc(key, msg)
 return (r,c)
```

package	a collection of typed procedure implementations with shared state
interface	set of (typed) locations, 2 collections of (typed) procedure names: imports & exports
seq. comp. $P_1 \circ P_2$	inlining: replace call to imported procedure `x ← f(a)` in P_1 with `x ← P₂.f(a)`
prerequisites	provide all imports. No requirement about state!
par. comp. $P_1 \parallel P_2$	union of implementations
prerequisites	no clashing procedure names

SSProve/Packages

```
package:  IND-CPA^0
mem:  key :  option KEY
```

```
ENC(msg):
 if !key == ⊥ then
   k <$ uniform {0,1}^n
   key := k
 (r,c) ← enc(key, msg)
 return (r,c)
```

package	a collection of typed procedure implementations with shared state
interface	set of (typed) locations, 2 collections of (typed) procedure names: imports & exports
seq. comp. $P_1 \circ P_2$	inlining: replace call to imported procedure `x ← f(a)` in P_1 with `x ← ` P_2`.f(a)`
prerequisites	provide all imports. No requirement about state!
par. comp. $P_1 \parallel P_2$	union of implementations
prerequisites	no clashing procedure names

Laws:

$$\begin{aligned} P_1 \circ (P_2 \circ P_3) &= (P_1 \circ P_2) \circ P_3 \\ P_1 \parallel P_2 &= P_2 \parallel P_1 \\ P_1 \parallel (P_2 \parallel P_3) &= (P_1 \parallel P_2) \parallel P_3 \\ (P_1 \circ P_3) \parallel (P_2 \circ P_4) &= (P_1 \parallel P_2) \circ (P_3 \parallel P_4) \end{aligned}$$

Games, Adversaries, Indistinguishability

- **Game**: a package with no imports
- **Game pair**: two games with the same exports
- **Adversary** $\mathcal{A}$ for game G: package compatible with G **with separate state** exporting one procedure

  $\mathcal{A}.\texttt{run} : \texttt{unit} \to \texttt{bool}$

  intuitive meaning: guess which game $\mathcal{A}$ is interacting with

Games, Adversaries, Indistinguishability

- **Game**: a package with no imports
- **Game pair**: two games with the same exports
- **Adversary** $\mathcal{A}$ for game G: package compatible with G **with separate state** exporting one procedure

  $\mathcal{A}.\texttt{run} : \texttt{unit} \to \texttt{bool}$

  intuitive meaning: guess which game $\mathcal{A}$ is interacting with
- **Advantage** of $\mathcal{A}$ against a game pair (G_0, G_1):

$$\alpha_{(G_0,G_1)}(\mathcal{A}) = |\Pr[\mathsf{true} \leftarrow (\mathcal{A} \circ G_0).\mathit{run}()] - \Pr[\mathsf{true} \leftarrow (\mathcal{A} \circ G_1).\mathit{run}()]|$$

- Perfect **indistinguishability** $G_0 \overset{0}{\approx} G_1 : \quad \forall \mathcal{A}\,.\ \alpha_{(G_0,G_1)}(\mathcal{A}) = 0$

SSP theorems

Theorem (Triangle inequality)

$\alpha_{(F,H)}(\mathcal{A}) \leq \alpha_{(F,G)}(\mathcal{A}) + \alpha_{(G,H)}(\mathcal{A}).$

SSP theorems

Theorem (Triangle inequality)

$\alpha_{(F,H)}(\mathcal{A}) \leq \alpha_{(F,G)}(\mathcal{A}) + \alpha_{(G,H)}(\mathcal{A})$.

Theorem (Reduction)

$\alpha_{(M \circ G^0, M \circ G^1)}(\mathcal{A}) = \alpha_{(G^0, G^1)}(\mathcal{A} \circ M)$.

SSP theorems

Theorem (Triangle inequality)

$\alpha_{(F,H)}(\mathcal{A}) \leq \alpha_{(F,G)}(\mathcal{A}) + \alpha_{(G,H)}(\mathcal{A}).$

Theorem (Reduction)

$\alpha_{(M \circ G^0, M \circ G^1)}(\mathcal{A}) = \alpha_{(G^0, G^1)}(\mathcal{A} \circ M).$

Theorem (Relational equivalence $\implies$ perf. indistinguishability)

Two games are perfectly indistinguishable if all their procedures are (i) equivalent in the pRHL, and (ii) maintain a stable invariant on the game state.

Summary – SSProve:

A foundational	built on standard mathematical foundations
framework	code, packages, laws, pRHL, semantics, tactics
for modular	programs composed from packages
crypto proofs	security properties of probabilistic, stateful language
in Coq	mature proof assistant with clearly delimited TCB
Docs, code, info	`github.com/SSProve`