Mechanizing State Seperation for Modular Cryptographic Proofs

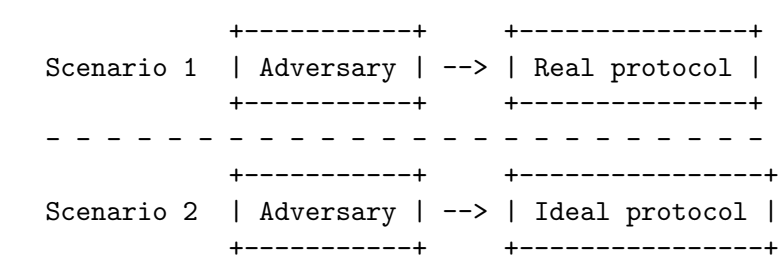
Markus Krabbe Larsen (krml@itu.dk)
IT University of Copenhagen

Iris Workshop 2024

June 5, 2024

Security statements

Security statements about cryptographic protocols are often expressed through a pair of modules (also called games).



With what probability can an adversary distinguish if it is in scn. 1 or scn. 2?
We prove security by proving a bound on the distinguishability for all adversaries.

The problem with lack of separation

Before separation:

For all adversaries, when the adversary and the real protocol use disjoint sets of names and the and the ideal protocol use disjoint sets of names, then the probability to distinguish the real and ideal protocol is less than ε.

After separation:

For all adversaries, ~~when the adversary and the real protocol use disjoint sets of names and the and the ideal protocol use disjoint sets of names, then~~ the probability to distinguish real and ideal protoocl is less than ε.

Let us have a closer look by modelling the primitives.[1]

[1] As in "State Separation for Code-Based Game-Playing Proofs" by Brzuka, Delignat-Lavaud, Fournet, Kohbrok and Kohlweiss

What are modules made of?

We consider the following impure actions:

```
sample
get
put
call
```

Example (One-time pad): For some alphabet-size k,

```
fn init(m : Fin k) -> ():
  s <- sample {0,..,k-1}
  put secret s

fn encrypt(m : Fin k) -> Fin k:
  s <- get secret
  return (s + m)
```

Modules

Functions may be put together as modules.

```
Module: OTP (One-time pad)
State: secret

fn init(m : Fin k):
  s <- sample {0,..,k-1}
  put secret s

fn encrypt(m : Fin k):
  s <- get secret
  return (s + m)
```

We write $\Sigma(M)$ for the finite set of state variables of a module M.
Example: $\Sigma(\texttt{OTP}) = \{\texttt{secret}\}$.
State variables are supposed to be "module-scoped".

A pair of modules

These modules express the property of One-Time Secrecy.

```
Module: OTS Real
State: secret

fn query(m : Fin k):
  call init ()
  c <- call encrypt m
  return c
```

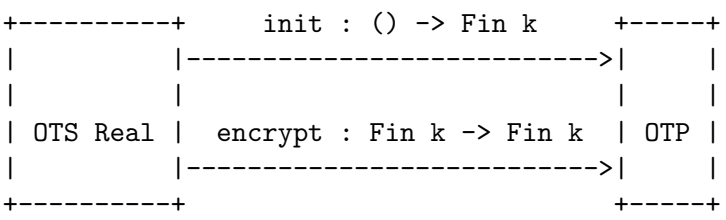
```
Module: OTS Ideal
State:

fn query(m : Fin k):
  c <- sample {0,..,k-1}
  return c
```

Notice that the interfaces are the same and that `OTS Real` expectes to call functions named `init` and `encrypt`.

Module composition

Modules are composed by replacing function calls with their definition. Thus we must consider overlapping state variable names.

$$\mathrm{OTS}_{\mathrm{real}} \circ \mathit{OTP}$$



In theory this is solved by "injectively renaming state variables", but this is not sufficient to base a machanization upon.
Currently, in SSProve state is shared.

Advantage

An adversary is a module that defines a function $\texttt{run} : () \to \texttt{bool}$.
For modules M, M' and a specific adversary A we define

$$\alpha(M, M')(A) := |\ \Pr_{\texttt{run}}(A \circ M \Rightarrow \texttt{true})\ - \Pr_{\texttt{run}}(A \circ M' \Rightarrow \texttt{true})\ |$$

Usually we want to consider all adversaries. For example:

$$\forall A,\ \Sigma(A) \cap \Sigma(\mathrm{OTP}) = \emptyset \to \alpha(\mathrm{OTS}_{\mathrm{Real}} \circ \mathrm{OTP}, \mathrm{OTS}_{\mathrm{Ideal}})(A) \leq 0$$

Indistinguishability

For modules M and M' define the notation

$$M \approx_0 M' := \forall A, \Sigma(A) \cap \Sigma(M) = \emptyset \to \Sigma(A) \cap \Sigma(M') = \emptyset \to \alpha(M, M')(A) = 0$$

If we can prove the following probabilistic relational hoare quadruple (in SSProve)

$$\begin{aligned}
&\{\lambda h\ h', h = h'\} \\
&\quad (\mathrm{OTS}_{\mathrm{Real}} \circ \mathrm{OTP})\ \mathrm{query}\ () \\
&\qquad \approx \\
&\quad (\mathrm{OTS}_{\mathrm{Ideal}})\ \mathrm{query}\ () \\
&\{\lambda (h, a)\ (h', a'), h = h' \wedge a = a'\}
\end{aligned}$$

we obtain

$$\mathrm{OTS}_{\mathrm{Real}} \circ \mathrm{OTP} \approx_0 \mathrm{OTS}_{\mathrm{Ideal}}$$

But what if we want to use the theorem?

Nominals

We let name permutations act on values of different types e.g code and modules.

$$\{\texttt{secret} \mapsto \texttt{foo}; \texttt{foo} \mapsto \texttt{secret}\} \cdot (\texttt{put secret } x) = (\texttt{put foo } x)$$

Some operators are equivariant e.g. for any permutation π and modules M and M',

$$\pi \cdot (M \circ M') = (\pi \cdot M) \circ (\pi \cdot M').$$

Nominals also gives us a natural notion of α-equivalence

$$M \equiv M' := \exists \pi, \pi \cdot M = M'$$

Notice, that module composition is not a congruence under α-equivalence.

Finding fresh names

The support of a value is the set of names that it "contains". For modules, the support is given by Σ.

If we are guaranteed an unlimited source of fresh names we can for $x : X$ and $y : Y$ define a permutation $\mathrm{fresh}\ x\ y$ such that

$$\mathrm{supp}\ x \cap \mathrm{supp}\ (\mathrm{fresh}\ x\ y \cdot y) = \emptyset$$

Separated composition

Use fresh to ensure separation of modules

$$M \circledast M' := M \circ (\mathrm{fresh}\ M\ M' \cdot M')$$

Equalities are preserved, however under alpha equivalence

$$M \circledast (M' \circledast M'') \equiv (M \circledast M') \circledast M''$$

Importantly, separated composition is a congruence w.r.t alpha equivalence

$$M \equiv M' \Rightarrow N \equiv N' \Rightarrow M \circledast N \equiv M' \circledast N'$$

Compatability

For any module M and permutaiton π,

$$M \approx_0 \pi \cdot M$$

For any modules M and M',

$$M \equiv M' \to M \approx_0 M'$$

For any modules M and M',

$$\Sigma(M) \cap \Sigma(M') = \emptyset \to M \circ M' \equiv M \circledast M'$$

Separated advantage

Advantage can now be defined in terms of separated composition

$$\alpha^*(M, M')(A) :=\mid \Pr_{\texttt{run}}(A \circledast M \Rightarrow \texttt{true}) - \Pr_{\texttt{run}}(A \circledast M' \Rightarrow \texttt{true}) \mid$$

which gives name-agnostic theorems

$$M \approx_0 M' \Rightarrow \forall A, \alpha^*(M, M')(A) = 0$$

while renaming does not affect the actual advantage

$$M \equiv M' \Rightarrow N \equiv N' \Rightarrow A \equiv A' \Rightarrow \alpha^*(M, N)(A) = \alpha^*(M', N')(A')$$

An abstract example

We show the security of some protocol Crypt.
Assume that we have previously shown, that

$$\mathrm{Crypt}_{\mathrm{Real}} \approx_0 \mathrm{Red} \circledast \mathrm{Foo}, \quad \mathrm{Crypt}_{\mathrm{Ideal}} \approx_0 \mathrm{Red} \circledast \mathrm{Bar}, \quad \forall A, \alpha^*(\mathit{Foo}, \mathit{Bar})(A) \leq \tfrac{3}{7}$$

Let A be an arbitrary adversary, then

$$\begin{aligned}
&\alpha^*(\mathrm{Crypt}_{\mathrm{Real}}, \mathrm{Crypt}_{\mathrm{Ideal}})(A) \\
&\leq \alpha^*(\mathrm{Crypt}_{\mathrm{Real}}, \mathrm{Red} \circledast \mathrm{Foo})(A) + \alpha^*(\mathrm{Red} \circledast \mathrm{Foo}, \mathrm{Crypt}_{\mathrm{Ideal}})(A) \\
&\leq 0 + \alpha^*(\mathrm{Red} \circledast \mathrm{Foo}, \mathrm{Crypt}_{\mathrm{Ideal}})(A) \\
&\leq \alpha^*(\mathrm{Red} \circledast \mathrm{Foo}, \mathrm{Red} \circledast \mathrm{Bar})(A) + \alpha^*(\mathrm{Red} \circledast \mathrm{Bar}, \mathrm{Crypt}_{\mathrm{Ideal}})(A) \\
&\leq \alpha^*(\mathrm{Red} \circledast \mathrm{Foo}, \mathrm{Red} \circledast \mathrm{Bar})(A) + 0 \\
&\leq \alpha^*(\mathrm{Foo}, \mathrm{Bar})(A \circledast \mathrm{Red}) \\
&\leq \tfrac{3}{7}
\end{aligned}$$

Case study: KEM-DEM

Using the KEM-DEM paradigm we can obtain general results about our hybrid encryption scheme.

```
Theorem PKE_security :
  ∀ {LA}
    (A : nom_package),
    ValidPackage LA PKE_CCA_out A_export A →
    AdvantageP (PKE_CCA KEM_DEM) A <=
    AdvantageP KEM_CCA (dlink A
      (dlink (MOD_CCA KEM_DEM) (dpar (nom_ID KEM_out) (DEM true)))) +
    AdvantageP DEM_CCA (dlink A
      (dlink (MOD_CCA KEM_DEM) (dpar (KEM false) (nom_ID DEM_out)))) +
    AdvantageP KEM_CCA (dlink A
      (dlink (MOD_CCA KEM_DEM) (dpar (nom_ID KEM_out) (DEM false)))).
Proof.
  intros LA A VA.
  unfold AdvantageP.
  erewrite (AdvantageD_perf_l (PKE_CCA_perf true)).
  erewrite (AdvantageD_perf_r (PKE_CCA_perf false)).
  rewrite 2!Aux_AuxD /AuxD.
  rewrite AdvantageD_dlink.
  eapply le_trans.
  + eapply (single_key_b _ _ _ (KEM false)).
    all: dprove_valid.
  + rewrite 3!dlink_assoc //.
Qed.
```

Case study: Zero Knowledge OR-proof (Work in progress)

The property of Special Honest Verifier Zerk Knowledge is expressed through the two modules Real and Ideal.
For OR-proofs we want to show that when

$$\forall A,\ \alpha^*(\mathrm{Real}(P), \mathrm{Ideal}(P))(A) \leq \varepsilon_1$$

$$\forall A,\ \alpha^*(\mathrm{Real}(Q), \mathrm{Ideal}(Q))(A) \leq \varepsilon_2$$

then

$$\forall A,\ \alpha^*(\mathrm{Real}(OR\ P\ Q), \mathrm{Ideal}(OR\ P\ Q))(A) \leq \varepsilon_1 + \varepsilon_2$$

part of which follows by from the fact that there is a module Call such that

$$\mathrm{Real}(OR\ P\ Q) \approx_0 \mathrm{Call} \circledast ((\mathrm{Patch}_L \circledast \mathrm{Real}(P)) \ddagger (\mathrm{Patch}_R \circledast \mathrm{Real}(Q)))$$

Incidentally, this argument also marks the place where I entered the rabbit hole of dealing with names.

Thanks!