CRYPTOGRAPHIC PROOFS IN APPROXIS

ZACHARIE MOUGHANIM AND SUPERVISED BY LARS BIRKEDAL AND PHILIPP HASELWARTER

Abstract. Security proof of protocols are naturally formulated as equivalences between systems, however, these equivalences need not be exact, we only require that they remain true up to a negligible error. We talk about computational security (as opposed to unconditional security), instead of perfect secrecy. Namely, that an attacker with *reasonable* computation time cannot crack the protocol within a *reasonable* error.

The Approxis program logic provides a framework to prove equivalences between probabilistic programs up to an error, therefore allowing us to consider stating and proving such computational guarantee within the Approxis logic.

In this report, we express in the Approxis logic several cryptographic properties, and write different known cryptographic proofs. We propose a way to model two distinct kinds of attacker on a protocol, prove the security of a protocol example in one of these modelisations, and finally explore ways to complete the same proof for a stronger, more realistic, attacker.

1. Introduction

Since Shannon's theorem it is known the right aim for security is not perfect secrecy, but rather computational secrecy: we only need to show that a cryptographic system is secure for any reasonable attacker i.e. polynomial-time program, and only up to a negligible error i.e. smaller than the inverse of any polynomial.

Approxis [1] is a higher-order relational separation logic, mechanized using the Iris [2] logic. The program equivalence aimed in Approxis is the *contextual equivalence*, namely the equivalence of programs when plugged in any context. However, these equivalences are hard to prove directly e.g. by induction on a context. We thus often use an intermediate, stronger (but easier to prove) notion, such as bisimulation, denotational semantics or *logical relations* [3], this last one being precisely the notion chosed for Approxis. Approxis provides a variety of tactics to prove logical relations between *probabilistic* programs up to an error, written in a ML-like probabilistic language, RandML. Like other probabilistic relational logics, it makes use of *probabilistic couplings*: while proving a relation between two programs, we can establish how the probabilistic samplings of both programs are related.

Since cryptographic primitives like encryption, hash function etc can often be straightforwardly written as a probabilistic language, there are quite simple to express in the RandML language. Stating computational cryptographic guarantees on these primitives is therefore made simple. Before the internship, several properties on primitives were already shown in Approxis, for instance IND-CPA$ security of a PRF-based encryption scheme, and of the ElGamal encryption scheme.

These two first examples rely on concrete primitives, but the proof of PRF already emphasize on one of the most important idea of cryptographic proofs: this proof is modular, relying on the construction of an *abstract* PRF. Therefore, the proof is rather a *reduction* of the IND-CPA$ assumption to the PRF assumption.

In Section 6, we present the first contribution of the internship: we prove IND-CPA$ security of a KEMDEM [4] scheme based upon a one-time secure symmetric encryption scheme and a IND-CPA$ asymmetric encryption scheme. This proof relies on *abstract* schemes, as aforesaid, moreover, the abstraction allows the schemes to be *stateful*. Even though it is never the case in real-life schemes for it would imply to have a secret shared memory between distant machines, it is often used to model an idealized primitive. For instance, the idealized PRF previously defined in Approxis strongly relies on an internal map.

Nonetheless, a cryptographic proof goes beyond proving equivalences between primitives, the ultimate foal is rather proving equivalences between *protocols*. A protocol can be seen as a sequence of *messages* exchanged over a *network* between different *machines*, each machine can react to the messages received. A protocol is more intricate than a primitive, and modelling a protocol as a probabilistic program thus requires more work. We need to decide how to model a *network*, and most of all, how to model the power of the attacker on said network.

In Section 7, we study two different classic modelisation of a network and the associated attacker.

- A network with eavesdropping attacker: the attacker can simply listen to each message sent over the network, but cannot intervene, they cannot modify a message;
- a network with active attacker: the attacker has access to all messages sent over the network. They can modify them, simply block them and even send them to someone else than their destinators.

We propose a way to model these two kind of attackers on a protocol example, Wide-Mouthed Frog [5], requiring only a symmetric encryption scheme. We prove security of the protocol against an eavesdropping attacker under the IND-CPA$ assumption on the encryption scheme. We finally explore how to complete a similar security proof against an active attacker. This last proof is highly more intricate, relying on more complex cryptographic games. We therefore introduce two *integrity* games, and we implement the simpler one in Approxis. We conclude by listing leads to conclude this proof.

2. Liminaries

2.1. **Separation logic.**

We consider programs interacting with a heap. For a logic to reason about these programs, we need a different notion of conjunction:

Separating conjunction: In addition to the classical conjunction $\cdot \wedge \cdot$, we introduce $P * Q$, which means we can split the heap in a part satisfying P and a part satisfying Q.

Points-to predicate: We have a new predicate: $\cdot \mapsto \cdot$ to state a location on the heap currently stores a given value.

Hence, for any location l and value v, we have that $l \mapsto v * l \mapsto v$ cannot be satisfied by any heap fragment, even though $l \mapsto v \wedge l \mapsto v$ is equivalent to $l \mapsto v$.

2.2. **Iris.**

The Iris logic [2] is a framework for higher-order concurrent separation logic, fully implemented in the Coq/Rocq proof assistant. All our proofs will be conducted in the *Iris Proof Mode*.

2.3. **Program equivalence.**

We want to study equivalence of programs. The notion we chose is *contextual equivalence*, namely, two programs are contextually equivalent: $e_1 \cong e_2$ when for every context C (a program with a hole), if we plug these programs in this context, there is a similar "observable behavior" between $C[e_1]$ and $C[e_2]$.

Here, the "observable behavior" will be termination. Thus, we'll say two programs e_1, e_2 are contextually equivalent when for every context C, $C[e_1]$ terminates if and only if $C[e_2]$ terminates.

Example 1: This may seem a rather loose requirement, but it is actually very restrictive, because of the quantification over any contexts. For instance, it is straightforward to see that contextual equivalence implies that two programs always evaluate to the same value.

Take two contextually equivalent programs returning an integer. For any integer n, consider the context: $\mathsf{if}\ \square = n\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{loop}$ where "loop" is a non terminating program. By contextual equivalence, we get that the two programs always evaluate to the same integer (if they terminate).

Actually, we won't directly prove program equivalence but rather *program refinements*: we say $e_1 \lesssim e_2$ whenever for every context C, if $C[e_1]$ terminates, then so does $C[e_2]$.

2.3.1. *Logical relations.*

Logical relations [3] describe proof techniques for several properties about programs. Here, we use these proof techniques to prove contextual equivalence, since contextual equivalence is a property really hard to prove directly, because of the universal quantification over all possible contexts.

The main idea of logical relations is that for two related programs, from related inputs, we get related outputs".

2.4. **Clutch.**

We briefly describe Clutch [6], the logic allowing to prove logical refinements of probabilistic programs.

2.4.1. *RandML language.*

We consider an expressive language, with references, fixpoint, existential, universal and recursive datatypes, and random sampling. We refer the reader to the introduction paper to Clutch [6] for a more thorough discussion of this language.

$$
\begin{aligned}
\mathit{Val} \ni v, w, ... &::= z \mid b \mid () \mid l \mid \iota \mid \mathsf{rec}\ f\ x = e \mid \langle v, w\rangle \mid \mathsf{inl}(v) \mid \mathsf{inr}(v) \\
\mathit{Expr} \ni e, e', e''... &::= v \mid x \mid (e)e' \mid \mathsf{if}\ e\ \mathsf{then}\ e'\ \mathsf{else}\ e'' \mid \mathsf{fst}(e) \mid \mathsf{snd}(e) \mid \langle e, e'\rangle \mid \\
&\quad \mathsf{ref}(e) \mid !e \mid e := e' \mid \mathsf{match}\ e\ \mathsf{with}\ \mathsf{inl}(v) \to e'; \mathsf{inl}(w) \to e' \mid \\
&\quad \mathsf{fold}(e) \mid \mathsf{unfold}(e) \mid \Lambda e \mid e\_ \mid \mathsf{pack}\ e \mid \mathsf{unpack}\ e\ \mathsf{as}\ x\ \mathsf{in}\ e' \mid \\
&\quad \mathsf{tape}(e) \mid \mathsf{rand}(e, e') \mid e + e' \mid ... \\
t, ... \in \mathit{Tape} &\overset{\text{def}}{=} \left\{(N, \vec{n}) \mid N \in \mathbb{N}, \vec{n} \in (\mathbb{N}_{\leq N})^*\right\} \\
\sigma, ... \in \mathit{State} &\overset{\text{def}}{=} \left(\mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val}\right) \times \left(\mathit{Label} \xrightarrow{\text{fin}} \mathit{Tape}\right) \\
\rho, ... \in \mathit{Cfg} &\overset{\text{def}}{=} \mathit{Expr} \times \mathit{State} \\
\mathit{Type} \ni \tau, \tau', ... &::= \alpha \mid \tau \to \tau' \mid \tau \times \tau' \mid \tau + \tau' \mid \forall x.\tau \mid \exists x.\tau \mid \mathsf{ref}\ \tau \mid \mathsf{tape}
\end{aligned}
$$

Where $z \in \mathbb{Z}, b \in \mathbb{B} \overset{\text{def}}{=} \{\mathsf{true}, \mathsf{false}\}, \iota \in \mathit{Label}$, x (resp. α) varies over a set of term variables (resp. of type variables). Expressions, values and types are considered up to α-equivalence. The set $(\mathbb{N}_{\leq N})^*$ stands for all finite sequence of naturals less than or equal to N.

FIGURE 1. The RandML language

We will write $\lambda x.e$ to denote $\mathsf{rec}\ \_\ x = e$ i.e. recursive functions that do not contain any recursive call, and $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ for $(\lambda x.e_2)e_1$.

- *Tape*: presampling tapes are proof devices, that allow to pre-load randoms sampled later in the program. Even though they appear syntactically in the language, they have no consequence in the final results. We will see how they intervene in the behavior of programs when we describing the operational semantics.
- *State*: a state has to keep track of every location and the values they point to, as for a language without randoms, but also every tape locations and the state of each tapes.
- *Cfg*: a configuration is a pair of an expression (a program currently running) and a state, describing the values pointed to by the references and the values stored in the tapes.

Example 2: For instance, we can write: $\mathsf{let}\ r := \mathsf{rand}(N)\ \mathsf{in}$
$\mathsf{if}\ r = N\ \mathsf{then}$
  $N + 1$
else
  r
a simple program sampling a random in $\{0, 1, ..., N-1, N+1\}$.

2.4.1.1. *Operational semantics.*

The operational semantics is mainly standard, since we're working with randoms primitives, it is not a deterministic relation, but rather a distribution over all possible next step. The only peculiarity is the usage of presampling tapes.

Random sampling $\mathsf{rand}(N)$ reduces to i where $i \in \{0, ..., N\}$ with probability $\frac{1}{N+1}$.

However, to reduce $\mathsf{rand}(\iota, N)$ (random sampling from a presampling tape), the behavior changes depending on the content of the tape:

- if the tape is empty, it behaves as $\mathsf{rand}(N)$;
- if the tape contains at least one element, we pop the head and return it, the reduction is thus deterministic.

Definition 1. (RandML reduction):

- Pure, deterministic reductions:
  - $(\mathsf{rec}\ f\ x = e_1)e_2 \overset{\text{pure}}{\rightsquigarrow} e_1[\mathsf{rec}\ f\ x = e_1/f][e_2/x]$;
  - ...
- Reductions with effects:
  - $(\mathsf{ref}\ v, \sigma) \to (l, \sigma\{l \mapsto v\})$;
  - $(l := w, \sigma\{l \mapsto v\}) \to ((), \sigma\{l \mapsto w\})$;
  - $(!l, \sigma\{l \mapsto v\}) \to (v, \sigma\{l \mapsto v\})$.
- Random sampling and tapes:
  - $(\mathsf{rand}(N), \sigma) \to \begin{cases} (0,\sigma) \text{ with probability } \frac{1}{N+1} \\ \dots \\ (N,\sigma) \text{ with probability } \frac{1}{N+1} \end{cases}$;
  - If $\sigma(\iota) = (N, [])$, then $(\mathsf{rand}(\iota, N), \sigma) \to \begin{cases} (0,\sigma) \text{ with probability } \frac{1}{N+1} \\ \dots \\ (N,\sigma) \text{ with probability } \frac{1}{N+1} \end{cases}$;
  - If $\sigma(\iota) = (N, [n_1, n_2, ..., n_k])$ $(\mathsf{rand}(\iota, N), \sigma) \to (n_1, \sigma')$ where $\sigma'(\iota) = (N, [n_2, ..., n_k])$ and is equal to σ on all other inputs.
  - $(\mathsf{tape}\ N, \sigma) \to (\iota, \sigma\{\iota \mapsto (N, [])\})$ where ι is a fresh label in σ.

Example 3: If a tape ι stores the following values : $(1, 4)$, the following programs: $\mathsf{rand}(\iota, 5) + \mathsf{rand}(\iota, 5) + \mathsf{rand}(\iota, 5)$ reduces to $5 + i$ with probability $\frac{1}{6}$ for each $i \in \{0, ..., 5\}$

Definition 2.: Let $\mathrm{exec}_{\Downarrow}(\rho)$ be the probability that ρ terminates.

Definition 3. (Typing): The typing judgement $\Theta \mid \Gamma \vdash e : \tau$ is mainly standard, and the important rules can be found in the paper introducing Clutch [6].

Definition 4. (Well typed context): A context C is well-typed, written $C : (\Theta \mid \Gamma \vdash \tau) \Rightarrow (\Theta' \mid \Gamma' \vdash \tau')$, if for every expression e such that $\Theta \mid \Gamma \vdash e : \tau$, then we have $\Theta' \mid \Gamma' \vdash C[e] : \tau'$.

Definition 5. (Contextual equivalence): We define contextual equivalence under Environment $\Theta \mid \Gamma$ and type τ as, for any well-typed context C, the probability that $C[e_1]$ terminates is bounded by the probability that $C[e_2]$ terminates:

$$\Theta \mid \Gamma \vdash e_1 \underset{\text{ctx}}{\precsim} e_2 : \tau \overset{\text{def}}{=} \forall \tau', (C : (\Theta \mid \Gamma \vdash \tau') \Rightarrow (\emptyset \mid \emptyset \vdash \tau)), \sigma,$$
$$\mathrm{exec}_{\Downarrow}(C[e_1], \sigma) \leq \mathrm{exec}_{\Downarrow}(C[e_2], \sigma)$$

2.4.1.2. *Relational reasoning.*

Clutch allow to prove *logical refinements*, an example of logical relations as desribed in Section 2.3.1, to prove contextual refinement via an adequacy theorem.

Since we're working with probabilistic program, the property is not directly termination, but rather the *probability of termination*. A contextual refinement $e_1 \lesssim e_2$ means that for all $\sigma \in State$, $\text{exec}_{\Downarrow}((e_1, \sigma)) \leq \text{exec}_{\Downarrow}((e_2, \sigma))$.

2.5. **Approxis.**

Approxis [1] is a logic allowing to prove logical refinements, and thus, contextual refinements, but *up to an error*. We can own a new kind of resource: *error credits* $↯\ \varepsilon$, that states that we can "spend" up to ε error to show our refinements.

Example 4: Let's say we want to show that $\mathsf{rand}(N)$ is equivalent to

$\mathsf{let}\ r := \mathsf{rand}(N)\ \mathsf{in}$
$\mathsf{if}\ r = N\ \mathsf{then}$
  $N + 1$
else
  r

We can choose to spend $↯\ \frac{1}{N+1}$ to avoid samplping N in this last program, and thus complete our proof.

3. The security assumptions

The goal of the internship was to write cryptographic proofs in Approxis. These proofs often rely on security assumptions and reduction, we describe here the assumptions we considered during the internship.

All the games presented below have the same form, we provide two libraries, whose functions will be called *oracles* to match the terminology of cryptography. We say a scheme satisfy the game when the two libraries are *indistinguishable*, in our case, we'll prove a contextual refinement between the two libraries.

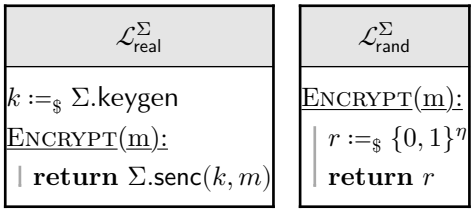
We have two distinct kind of security assumption: *indistinguishability* and *integrity*. These two are complementary and provide radically different guarantee on a scheme.

3.1. **Indistinguishability.**

Consider, for instance, an encryption scheme, indistinguishability states observing ciphers produced by this scheme does not provide information about the plaintext encrypted.

We consider two different indistinguishability properties, *indistinguishability against chosen plaintext attacks* (IND-CPA) and *indistinguishability against chosen ciphertexts attacks* (IND-CCA).

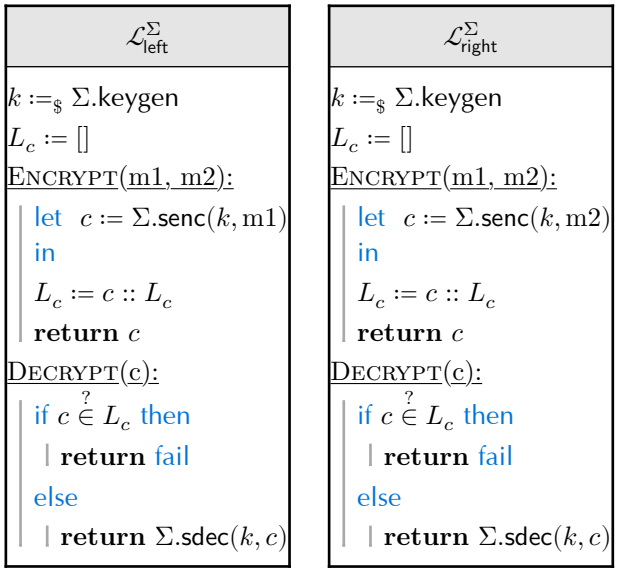
3.2. **IND-CPA$.**



GAME 1.. IND-CPA$ game for symmetric encryption

Game 1 defines the two libraries $\mathcal{L}^{\Sigma}_{\mathsf{real}}$ and $\mathcal{L}^{\Sigma}_{\mathsf{rand}}$, we thus observe that if these two libraries are indistinguishable, then the ecnryption of a plaintext m gives just as much information as a random sequence of bit i.e. none with overwhelming propbability. This assumption is actually stronger than what is usually called IND-CPA [4, Claim 15.3], but all the scheme known to satisfy IND-CPA that we studied in Approxis satisfy IND-CPA$ too.

3.3. **IND-CCA.**



GAME 2.. IND-CCA game for symmetric encryption

Game 2 defines the two libraries for the IND-CCA game. This is very similar to IND-CPA, but the attacker can additionnally decrypt messages instead of just encrypting. Here, we use the standard version rather than the random one. The attacker provides two plaintexts, and the game answers with a ciphertext corresponding to one or the other, but the attacker doesn't know which one.

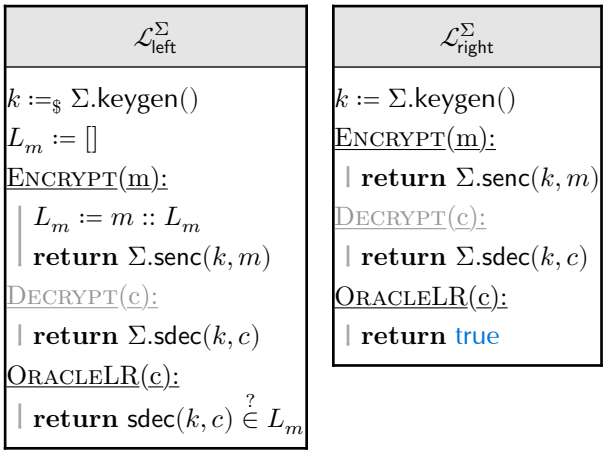
Notice that we record encrypted plaintexts to avoid decrypting what we previously encrypted with ENCRYPT. If we authorized that, an attacker could simply ask to encrypt (m_1, m_2), then decrypt the result and check equality with m_1 then m_2 to distinguish the two libraries.

3.4. **Integrity.**

Integrity states that the encrypted messages cannot be tampered with to forge messages that were not honestly encrypted i.e. with the knowledge of the key.

We study two notions of integrity: *integrity of plaintexts* (INT-PTXT) and *integrity of ciphertexts* (INT-CTXT). For both of these notions, a version with and without a decryption oracle appears in the literature, in our development we add a decryption oracle in both games.

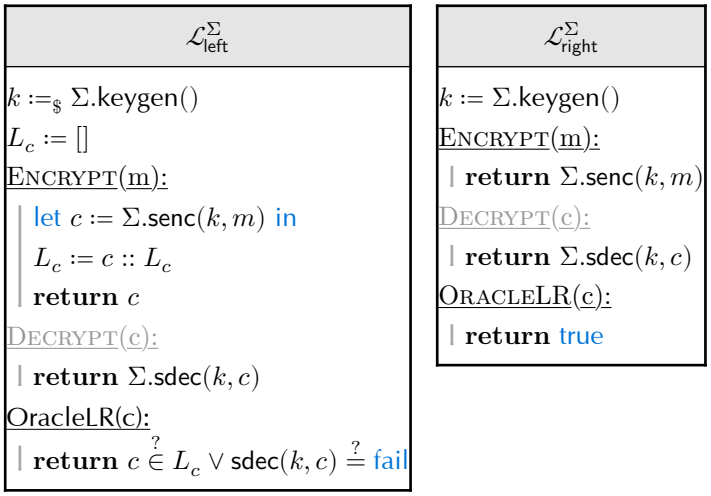
3.5. **INT-PTXT.**



Game 3.. Int-Ptxt game for symmetric encryption

Game 3 defines the libraries for the INT-PTXT game. It provides in both cases a perfect encryption and decrytion oracle, but the key oracle is OracleLR: on the left side, we test if the decrypted message was previously encrypted by the Encrypt oracle, whereas on the right side we answer true, which translates the fact that each valid cipher must have been encrypted by the oracle.

3.6. **INT-CTXT.**



Game 4.. Int-Ctxt game for symmetric encryption

Game 4 defines the libraries for the IND-CTXT game. This game is very similar to INT-PTXT, but here we check that each *ciphertext* was obtained via Encrypt. The nuance is that an attacker cannot modify an honest cipher to obtain another cipher of the same plaintext.

For instance, consider an encryption scheme ignoring the most significant bit of each plaintext (each ciphertext is prepended with 0 or 1 indifferently), then flipping the most significant bit provides another cipher, even though it wasn't obtained via an encryption function. Such an encryption scheme could satisfy INT-PTXT, but not INT-PTXT.

3.7. **Note on equivalences of security assumption.**

Let us briefly study the relations between each cryptographic assumptions. First of all, IND-CCA is a stronger assumption than IND-CPA, and INT-CTXT is stronger than INT-PTXT [7], but not for the same reason. Between the indistinguishability games, we add more oracle for the attacker, but the check remains more or less the same; whereas between the integrity games, we are simply more strict in the check performed: if a cipher is an output of Encrypt, then its decryption must be an input of Encrypt, but the converse need not be true.

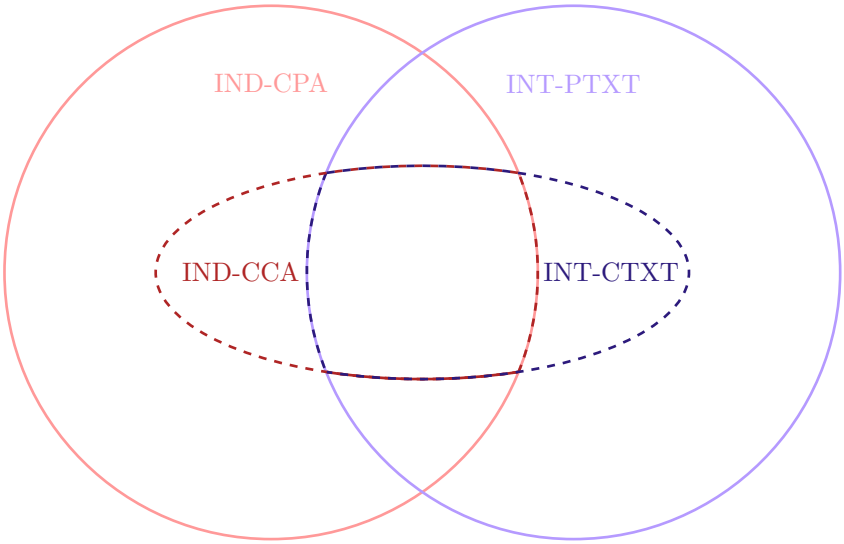


FIGURE 2. Comparison of cryptographic games

Figure 2 describes the interaction between indistinguishability and integrity. IND-CPA and INT-PTXT alone are completely independent, but if we assume at least IND-CPA and INT-PTXT, assuming either of the two other will provide the last one too.

4. State of the Art of cryptography in Approxis

Before the internship, there was already several files declaring definitions and theorems about cryptography, those relevant for the proof presented here are described below.

4.1. Symmetric encryption.

The file `symmetric.v` defines a typeclass for symmetric encryption scheme. As we'll see later, such a typeclass will allow to prove properties on abstract primitives.

4.2. Asymmetric encryption.

The file `pubkey.v` defines asymmetric encryption scheme, not as a typeclass though.

4.3. PRF based IND-CPA$ encryption.

A proof of CPA$-security for a PRF-based encryption scheme [4, Construction 7.4] can be found in `prf_cpa.v`.

In particular, this file idealizes the pseudo-random function to a function initializing a map, sampling random on uncalled inputs, and looking up the map when the function was already called on the input. This idealized PRF is therefore stateful, and extra care is required to treat such functions: this will be materialized by additional properties in an invariant, remembering what is stored in the map.

4.4. ElGamal asymmetric encryption scheme.

We also have an asymmetric encryption scheme satisfying CPA$-security, in `ElGamal_defs.v`, `ElGamal_semantic.v`, `ElGamal_syntactic.v` and `ElGamal_close_ctx.v`, we reduce the security assumption of ElGamal scheme [4, Construction 15.6] to DDH [4, Definition 14.5].

Unlike the PRF-based encryption of `prf_cpa.v`, this proof is a reduction: it does not idealize any underlying function, but rather rely on another equivalence.

5. Working with an opaque scheme

First, let's recall a scheme is defined as a key generation function, a pair of encryption and decryption (that can be initialized before usage, if the scheme is stateful), a function to generate random ciphers and its parameters.

Example 5: Consider the random function-based encryption. For correctness, the initialized map must be shared between the encryption and the decryption function.

If the map is not shared, then $F(k, r)$ will return two different values in the encryption and in the decryption with probability $\frac{N-1}{N}$, thus the scheme cannot be correct.

5.1. **Opaque initialization.**

This leads to think we need to consider that the scheme can be initialized when working with an opaque encyption scheme. The initialization is performed automatically when evaluating the field `enc_scheme` of the `SYM_init` typeclass. For this, we add some definitions:

- `P0l/P0r`: when finishing evaluation of the encryption scheme, we get a property stating "the scheme is initialized" and the encryption/decryption can thus be run safely.
- `Pl/Pr`: it's the same thing as their `0`s counterpart, but it's not necessarily initialized. This means after at least one usage of encryption or decryption, we have `Pl` or `Pr` as hypothesis instead of `P0l/r`.
- `Plr`: this means that we have both `Pl` and `Pr`, but additionally, this means some kind of synchronisation between the left and the right side.

Plus, we can choose to give up `P0l` and `P0r` to get `Plr` to "synchronise" the two sides just after initializing.

- We moreover need to have that `P0l ∗ P0r` entails `Plr` i.e. when initialized, the two sides are synchronised.
- `refines_init_scheme`: this allow to replace the initialization of a scheme by the pair of function `senc` and `sdec`, and get the `P0l/r` as hypothesis.

Example 6: With the random-function based scheme, the map is initially empty, `P0l` and `P0r` therefore states the presence of an empty map on the heap. `Pl` and `Pr` simply states that there is a map on the heap, with unknown content, but `Plr` crucially states that the map on the left and right side is actually the same one.

This leads to one major point of these stateful schemes: when initialized, the two maps (on each side) is empty, and are thus equal.

5.2. **Describing a cipher.**

All encryption schemes must satisfy *correctness*, namely that decrypting an encrypted message returns said message. We could simply require that `enc k dec`

`k msg` reduces to `msg`, but if we think of a protocol for instance, we might want to delay correctness in several settings: e.g. of an encryption sent over a network and decrypted on another machine.

Hence, when encoutering an encryption, instead of running through the encryption by hand, we want to apply a lemma that replaces the encryption by an opaque value `c`, and providing an additional hypothesis `sym_is_cipher ... msg c ...`. When we later encouter a decryption of `c`, we can apply this hypothesis `sym_is_cipher` instead of running through the program, and get back the message `msg`.

5.2.1. *Another attempt at describing a cipher.*

We could also have considered a way to directly replace `c` wherever we want in the program by `senc k msg`, but there are two main drawbacks:

- this new version is strictly stronger than `sym_is_cipher`, under the assumption that the scheme is correct.
- Plus, if we want to prove that this behaves correctly, at some point we have to reduce backwards. Indeed, `senc k msg` reduces to `c`, but not the other way around, which makes this proof much harder in practice.

We draw a more general lesson for writing refinements from this attempt: when we write refinement lemmas, we always need to go *forward* in the reduction of the program i.e. if we replace an expression `e1` by another `e2`, `e1` should reduce to `e2`.

6. KEMDEM

In this section, we prove security of a kemdem scheme (https://garbledcircus.com/kemdem/left-right).

More precisely, we prove that with an asymmetric encryption scheme satisfying CPA$ security, and a symmetric encryption scheme satisfying one-time CPA$ security, we can construct a kemdem scheme satisfying CPA$ security.

6.1. **Generic proof.**

The proof is generic. This means it can be instantiated with any pair of schemes satisfying some lemmas, described in several typeclasses in the `kemdem_hybrid_cpa_generic.v` file.

6.1.1. *Initialization.*

We thus work with opaque schemes, and particularly, we need to consider we could work with a symmetric scheme relying on initialization. We therefore strongly use the definitions discussed in Section 5.1.

Implementation detail 1: One of the only assumptions that may seem ad-hoc comes from the fact that we need to reorder several computation at some point. This leads to considering strong coupling assumptions on abstract function, more precisely, on `rand_cipher` for both schemes.

6.2. **Instances.**

We have at hand one asymmetric encryption scheme satisfying the IND-CPA assumption: ElGamal, and two symmetric schemes satisfying the One-Time-Secrecy assumption: the PRF based encryption and One-Time-Pad.

Implementation detail 2: To instantiate the generic proof, we need some assumptions over the translation between group element and integer to use ElGamal (the two functions `vg_of_int` and `int_of_vg`), which can be found in each instance's file.

6.2.1. *One-Time-Pad & ElGamal.*

This case is pretty simple, as the scheme does not require any initialization. Instantiating the required typeclasses is rather straightforward.

Implementation detail 3: Note however the assumptions on `vg_of_int` to avoid using the rejection sampler are stated as axioms in the OTP file.

6.2.2. *PRF based scheme & ElGamal.*

Here we strongly use the definitions and lemmas described in Section 5.1, so `Pl`/`Pr` can keep track of the state of the map, as explained in Example 6.

Implementation detail 4: We do not assume more properties on `vg_of_int`, and rather work with a rejection sampler, we admit a result to avoid tampering with stripping laters, but this is highly perfectible.

7. Protocols in Approxis

In this section we try to prove properties about cryptographic protocols in Approxis. The first question is how to model these protocols; unlike Squirrel [8] or other proof assistant for cryptography, Approxis does not have a built-in notion of protocol.

In the remaining of this section, we will use the example of the Wide-Mouthed Frog (WMF) protocol (https://lsv.ens-paris-saclay.fr/Software/spore/wideMouthedFrog.html), informally described below:
$A \to S : (A, \{B, N\}_{\mathrm{ka}})$
$S \to B : \{A, N\}_{\mathrm{kb}}$
where:

- A and B are agent, exchanging a secret nonce N;
- S is a server trusted by both parties, having access to symmetric keys to communicate with both agents.

7.1. **Security of a protocol.**

There are several notions of security for a protocol, but we'll here consider *strong secrecy*. We can clearly identify N as *the* value that must remain secret in the protocol it is a nonce i.e. a value obtained by random sampling, which is drawn exactly once through each run of the protocol. Strong secrecy states that the attacker cannot compute any bit of this nonce. Or equivalently that the attacker cannot differentiate between the protocol and the same protocol where N is replaced by a random known to the attacker.

7.2. **Modelling a protocol.**

The main difficulty is to give an account of the *network*, and in particular the power of the attacker upon the network.

Definitions of several variants of the protocol can be found in the file `wmf_protocol.v`.

7.2.1. *Eavesdropping attacker.*

Consider an attacker that can only read what passes through the network, but cannot intervene.

The first choice we make is to pass messages as arguments of an abstraction. The network is perfect in this sense: each message reaches its intended receiver and is not altered.

Then, we want to let the attacker access to every message exchanged on the network, we thus provide a tuple containing these messages at the end of the procol execution.

We prove security of the protocol against an eavesdropping attacker under the IND-CPA$ assumption over the symmetric encryption scheme in the file `wmf_eav_security.v`.

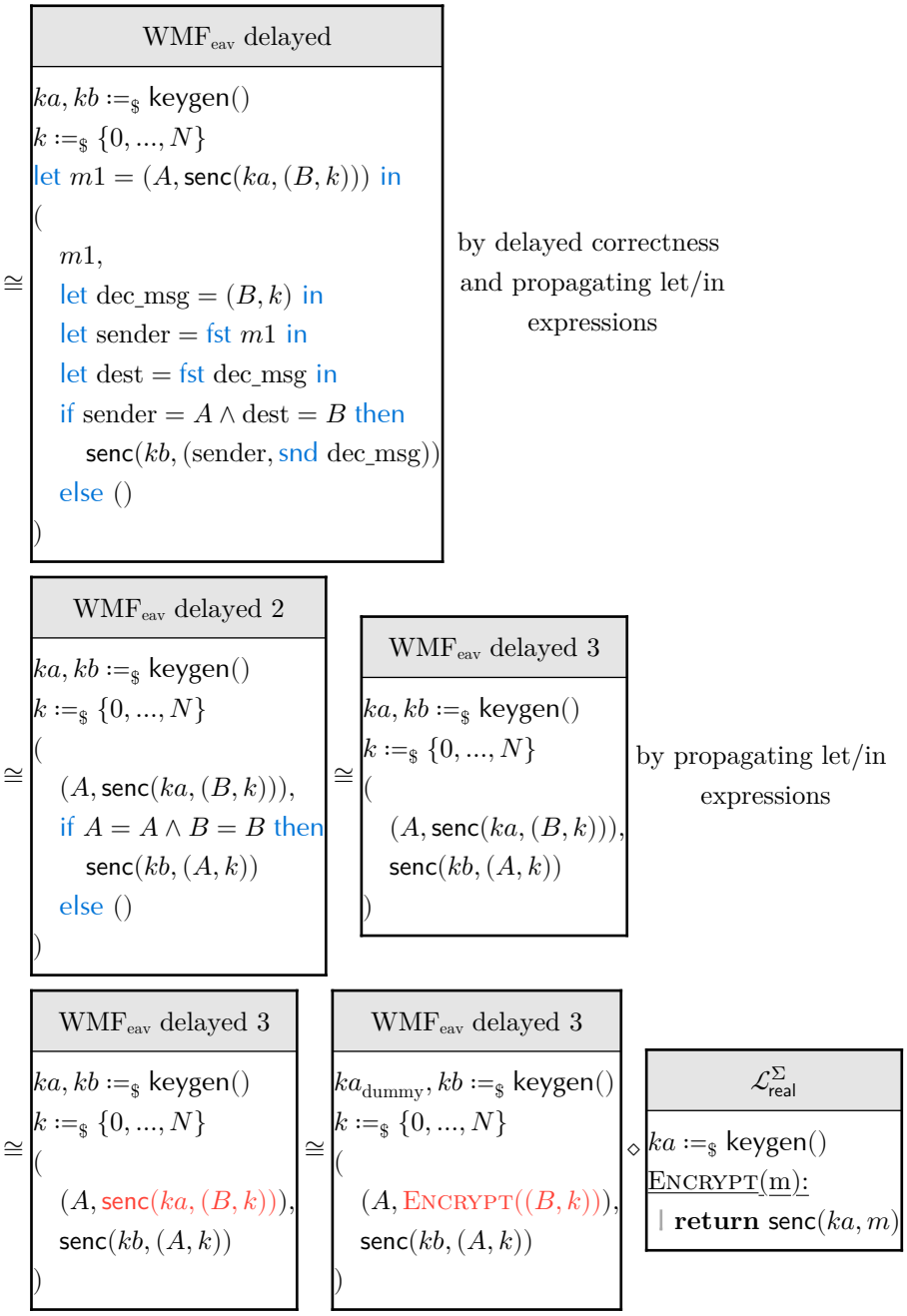
The key point of the proof is when we delay the encryption performed by agent A, to the code run by the server, that allows us to completely erase the decryption, and therefore use the CPA$ assumption, which does not provide a decryption oracle. This can only be done because the attacker only observes the network, we can fool the attacker, making them believe the nonce is sampled by A and sent, while it's simply sampled by S.

7.2.1.1. *Proof* à la *Rosulek.*

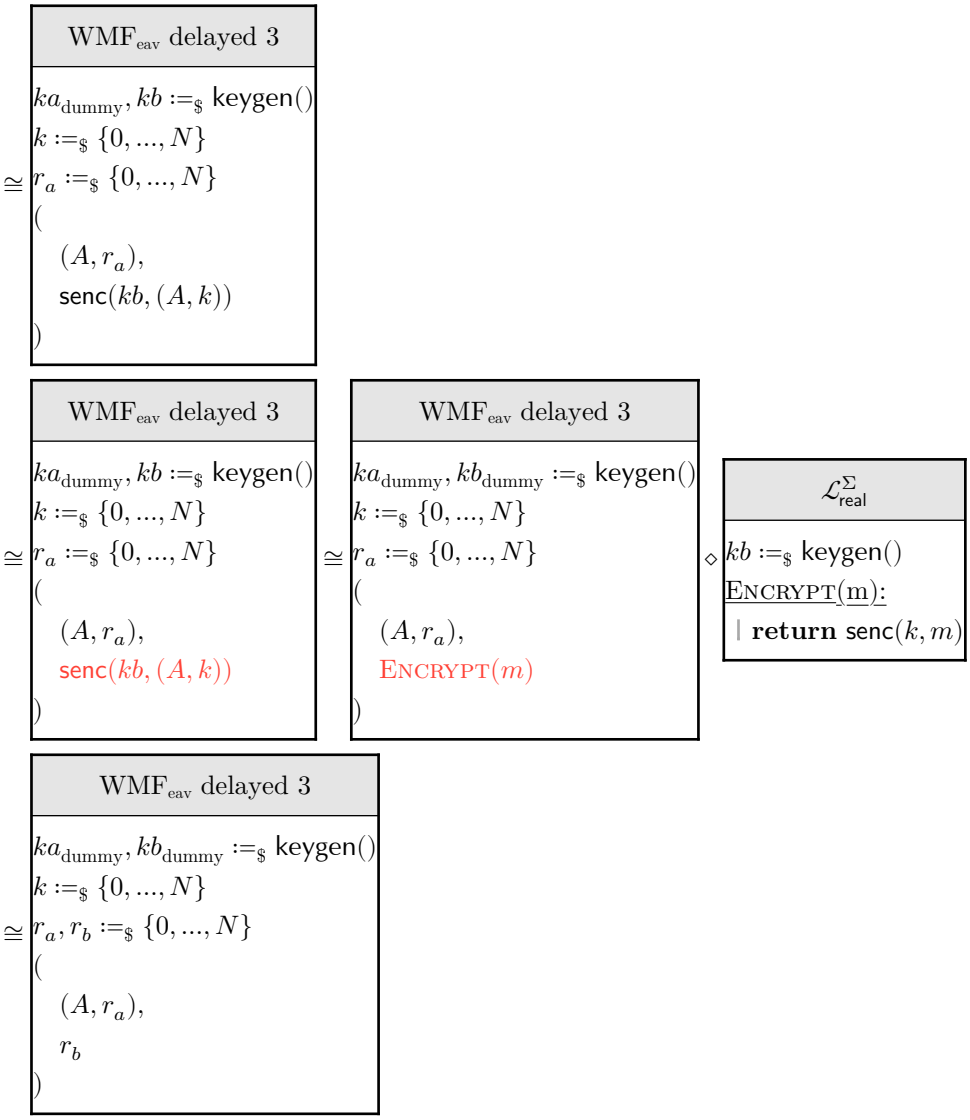
In this subsection, we present the equivalence step of the proof in the style of Mike Rosulek indistinguishability proofs. We reduce the protocol to the IND-CPA$ game (Game 1).

WMF_{eav}

$ka, kb :=_{\$} \mathsf{keygen}()$
$k :=_{\$} \{0, ..., N\}$
$\mathsf{let}\ m1 = (A, \mathsf{senc}(ka, (B, k)))\ \mathsf{in}$
(
  $m1$,
  $\mathsf{let}\ \text{dec\_msg} = \mathsf{sdec}(ka, \mathsf{snd}\ m1)\ \mathsf{in}$
  $\mathsf{let}\ \text{sender} = \mathsf{fst}\ m1\ \mathsf{in}$
  $\mathsf{let}\ \text{dest} = \mathsf{fst}\ \text{dec\_msg}\ \mathsf{in}$
  $\mathsf{if}\ \text{sender} = A \wedge \text{dest} = B\ \mathsf{then}$
  $\mathsf{senc}(kb, (\text{sender}, \mathsf{snd}\ \text{dec\_msg}))$
  $\mathsf{else}\ ()$
)



Notice in this last step that the sampling of the key for a becomes useless since it is not used, the actual key is now sampled within the oracle of the IND-CPA game.



And we can perform the same step the other way around to replace k by an adversarial random.

This proof displays the big steps of the proof available in `wmf_eav_security.v`, the implementation details are omitted.

7.2.2. *Active attacker.*

We now consider an attacker that has full control over the network.

To prove this we need an integrity property, because the attacker can send any message that will be decrypted by the server. Furthermore, we needed a decryption oracle in the indistinguishability game, because, unlike for the eavesdropping attacker, we cannot erase the decryption because we're not even sure that the server will decrypt the message encrypted by A. We chose to assume INT-PTXT and IND-CCA over the symmetric scheme. This turns out to be the wrong choice, the

INT-CTXT assumption being more useful. This is particularly damning due to the fact that INT-CTXT and IND-CCA is stricly equivalent to INT-PTXT and IND-CCA Figure 2. Nevertheless, the reason INT-PTXT was originally chosen is because it didn't require to add any more lemmas about maps, whereas implementing the INT-CTXT game would require general maps, not only mapping integers to values, as it is already the case in the `map.v` file.

The proof could not be carried out in the end, it can be found in the file `wmf_active_attacker_security.v`. To finish it, we must define the INT-CTXT game, and prove under this assumption we can replace decryption by looking up an associative map recording ciphers and plaintexts encrypted by the encryption oracle. For more details, see `intctxt_ideal_dec.v`. This last property could also be used as the definition of INT-CTXT. But, since the original definition is generic, we might need another alternative definition to prove a different protocol later, so implementing the actual definition of the game seems unavoidable.

Since the protocol encrypts with two different keys, all games used must provide the initialized primitives in addition to the oracles, specifically because of initialization. Otherwise, we could not synchronise encryption in refinements proofs with an adversary to the games, because we would have three schemes initialized and we can only synchronize two at the same time. If we were working with non-initialized scheme, we could just fetch the primitives from the typeclass and that would not be an issue. In particular, this wouldn't be an issue in a protocol using asymmetric encryption.

Implementation detail 5: We had to improve the `refines_senc` lemmas to carry out the proof here.

Recall in these lemma, we spend the `Plr/Pl/Pr` assumption to get `sym_is_cipher` instead. However, in a protocol with active attacker, we may never decrypt the encrypted message, but we may decrypt it too, depending on the behavior of the attacker. Therefore, when decrypting we need to be able to get both properties, but we can't get both at the same time. The tool for the job is the non-separating conjunction. We prove lemmas really similar to `refines_senc` but we get `Plr/Pl/Pr lls rls /\ sym_is_cipher lls rls c msg k` instead of simply `sym_is_cipher`, allowing to discharge the hypothesis and get the side corresponding to the behavior of the protocol.

7.2.3. *Modelling an attack.*

Another important feature is to see if Approxis is expressive enough to capture *attacks*. We weaken the WMF protocol to obtain an unsafe one, model and prove that it can be attacked if dishonest agents are involved in the file `wmf_attack.v`:
$A \to S : (A, B, \{N\}_{\mathrm{ka}})$
$S \to B : \{A, N\}_{\mathrm{kb}}$

The attack is an on-path attack, and can be informally described as:

- the attacker intercepts the message $(A, B, \{N\}_{\mathrm{ka}})$ and gives to S the message $(A, Bd, \{N\}_{\mathrm{ka}})$ where Bd is the tag of a dishonest agent.
- S sends to Bd the message $(A, \{N\}_{\mathrm{kbd}})$ and either the attacker can decrypt the nonce N or the dishonest agent can and provide it directly to the attacker via the network.

In our proof, we chose to let the dishonest agent decrypt the nonce and provide it to the network.

Moreover, we need $\frac{1}{2^\eta}$ error credits to complete the proof, where η is the length of the randoms. There is a $\frac{1}{2^\eta}$ chance that the secret nonce and the adversarial random are equal, and the attacker thus cannot distinguish the two sides of the protocol.

8. Conclusion

8.1. **Conclusion.**

We were able to prove several properties on some constructions based upon *abstract* primitives, and even allowing these primitives to be *stateful*, to work with idealized function using a shared state.

First, the KEMDEM construction could be proven secure in a fully generic way, and instantiated with two concrete pairs of scheme.

Then, the WMF protocol could be proven secure for a weak, eavesdropping attacker, once again in a fully generic way.

However, the protocol agains an active attacker was not proven secure, it requires more modification, specially of the map lemmas and the INT-CTXT game.

Plus, an attack on a modified, unsecure version of the same protocol could be modelled in Approxis.

8.2. **Future works.**

A first goal would be to instantiate the WMF protocol against eavesdropping attacker with a concrete scheme. We already have at hand a symmetric scheme satisfying the IND-CPA$ assumption needed for that with the PRF-based scheme.

Plus, even though the attack could be modelled in Approxis, an attack seems to be more of a unary property. We could consider proving the same thing in the Eris logic [9], which appears to be more appropriate.

Another more important goal would be to finish the security proof for the active attacker on WMF. However, this would require prior work: first, we need to write down the definition of the INT-CTXT game and if we want to use the definition described above, we need to extend the maps to maps from any type to any type (because the messages could be of type int as well as group elements). Then, we must prove that under INT-CTXT, we can replace decryption by a sequence of equality test to every cipher that was obtained beforehand. From this point on the proof could be finished if no unexpected problem came up.

The main issue in the active attacker proof was the maintenance of the enormous invariant in the active attacker proof. Basically, we need to keep track of whether each message is already sent and received. Since there is 3 such messages, there are $2^3 = 8$ possibilities. This is particularly frustrating when proving trivial branches of the proof, since we need to destruct the 8 different cases to perform trivial proof tactics on each of them. One major improvement point would be to find a way to reduce these time-consuming simple proofs. By definition, the properties in the invariant are formulas, and basically, a disjunction of more precise properties, describing each case. An invariant of the form of a transition system describing the protocol would be helpful, because most branch are actually alike.

Moreover, it becomes very hard to destruct and close invariants. It could be really helpful to be able to name some "atoms" (or we can think of states of the aforementioned transition system) of the invariant, and automatically destruct the proposition when opening an invariant.

Plus, the WMF protocol is rather small (only two messages and one reception) so the invariant for this protocol is a disjunction of 8 different cases, but if we want to prove more intricate protocols with Approxis, it will become really difficult to carry the proof with the current invariants.

For closing invariants, we could have an automatic program determining in which "state" of the invariant we are on, and closing it. Most of the invariants closure are similar and straightforward, but require to explore all 8 possibilities.

References

1. Approximate Relational Reasoning for Higher-Order Probabilistic Programs | Proceedings of the ACM on Programming Languages, https://dl.acm.org/doi/10.1145/3704877
2. JUNG, R., KREBBERS, R., JOURDAN, J.-H., BIZJAK, A., BIRKEDAL, L., DREYER, D.: Iris from the ground up: A modular foundation for higher-order concurrent separation logic. Journal of Functional Programming. 28, e20 (2018). https://doi.org/10.1017/S0956796818000151
3. Skorstengaard, L.: An introduction to logical relations
4. Rosulek, M.: The Joy of Cryptography, https://joyofcryptography.com/
5. Burrows, M., Abadi, M., Needham, R.: A logic of Authentication. (1989)
6. Gregersen, S.O., Aguirre, A., Haselwarter, P.G., Tassarotti, J., Birkedal, L.: Asynchronous Probabilistic Couplings in Higher-Order Separation Logic. Proceedings of the ACM on Programming Languages. 8, 753–784 (2024). https://doi.org/10.1145/3632868
7. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm, https://eprint.iacr.org/2000/025
8. Baelde, D., Delaune, S., Jacomme, C., Koutsos, A., Moreau, S.: An Interactive Prover for Protocol Verification in the Computational Model. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 537–554 (2021)
9. Aguirre, A., Barthe, G., Gaboardi, M., Garg, D., Katsumata, S.-y., Sato, T.: Higher-order probabilistic adversarial computations: categorical semantics and program logics. Proc. ACM Program. Lang. 5, (2021). https://doi.org/10.1145/3473598

ENS Rennes, Rennes, France

Aarhus University, Aarhus, Denmark