Composition in the Squirrel prover

Jules Timmerman

2024

Contents

Introduction

Machines all over the world use protocols to communicate information, for example for messaging or bank payments. This omnipresence intensifies the need for secure communication, thus requiring proof of security and of other properties like privacy. Cryptographers usually prove protocols secure using games and indistinguishability. They show that a protocol is indistinguishable from a version where security is obvious. This type of proofs uses the *computational model*, where the attacker is a Polynomial Probabilistic Turing Machine (PPTM). This model is close to what happens in reality, and reasoning often relies on probabilities: two randomly sampled keys may be equal, but the probability is really low. While manual proofs are possible, automated proofs scale better for bigger protocols and are less error-prone. Many tools can be used for mechanically proving protocols. Some are highly automated, like CryptoVerif, others require more user input, like Squirrel [3] or EasyCrypt [5].

When proving the security of a protocol, we often want to split the protocol in smaller parts. This allows us to reuse proofs, as well as dealing with simpler subproofs. For example, we want to easily prove security properties on various versions of SSH with different cipher suites running in parallel on the same machine. However, the notion of indistinguishability is hard to use as it is not composable most of the time. Indeed, shared secrets can be partially leaked by other protocols running in parallel, ruining the security. A few composition results have been established in the past. Here, we mostly focus on the results from [8]. The idea is the following: we first encapsulate the shared secrets' usage inside an oracle. We then prove security of other parts of the protocol *in presence of said oracle* – this is called $\mathcal{O}$-indistinguishability. If the oracle is powerful enough so that substituting the shared secret for the oracle in the protocol does not change the output, then you can deduce the security of the original protocol. This process of replacing part of a protocol by an oracle $\mathcal{O}$ strong enough is called $\mathcal{O}$-simulatability. Given the many advantages of composition, we would like to implement this result in Squirrel.

Squirrel uses bi-deduction to define cryptographic games and oracles [2]. In its simplest form, bi-deduction captures the idea that two indistinguishable terms stay indistinguishable when used inside an *adversarially computable* function. There are some similarities between $\mathcal{O}$-simulatability and bi-deduction, both stating the existence of some function computing terms using an oracle. This is what motivates us to use bi-deduction as a way to encode the composition result in Squirrel.

With the many advantages of composition, we wish to integrate the results from [8] inside Squirrel leveraging techniques from [2].

Contribution In this paper, we propose an extension of the current proof system of Squirrel to enable composition. With our additional rule, we are able to split protocols in multiple pieces. We can then prove security using the existing bi-deduction framework introduced in [8]. We provide a concrete example of how it could be used in practice.

Outline First we introduce the basics of the Squirrel logic in Section 1. Then, in Section 2, we detail the bi-deduction framework used in Squirrel as well as its ties to the composition result we will use. Lastly in Section 3, we motivate, state and prove a new inference rule for Squirrel and give an example on how it can be used.

Related Works Composition results have already been used in conjunction with automated tools. CryptoVerif used such composition result but it was limited to key exchange protocols [6]. You would prove the sub-proofs automatically with the tool but the composition theorem would

be used outside. EasyCrypt implemented proof mechanization using the UC framework [7]. This approach has the same caveats as UC: proving UC properties is sometimes too demanding as it requires the property to hold for any context. This bottom-up approach, that is starting from the primitives and going "up" to the general protocol is the opposite of what we do in this work.

Lastly, EasyCrypt also put an emphasis on a technique called State-Separating Proof [9]. This technique suggests a different way of writing games: instead of using an imperative style, you express games as a functional style. This allows doing proofs in a more modular way. This is not, however, a "real" composition result but more of a suggestion to ease proofs.

1 Squirrel

1.1 Computational indistinguishability in cryptography

In cryptography, computational indistinguishability expresses the idea that an attacker cannot differentiate two objects. Cryptographers show that a protocol is secure by proving that it is indistinguishable from an ideal, obviously secure version of it.

Games are a central notion in cryptography to formalize protocols. Consider two games $\mathcal{G}_I$ and $\mathcal{G}_R$. Given an attacker $\mathcal{A} \in \texttt{PPTM}$ and a game $\mathcal{G}$, we write $\mathcal{A}^{\mathcal{G}}$ the attacker interacting with the game. We say that two games are indistinguishable when:

$$\forall \mathcal{A} \in \texttt{PPTM}, \eta \mapsto |\mathbb{P}(\mathcal{A}^{\mathcal{G}_I} = 1) - \mathbb{P}(\mathcal{A}^{\mathcal{G}_R} = 1)| \text{ is negligible}$$

where being negligible means that the function is asymptotically smaller than the inverse of any polynomial, that is $\forall k \in \mathbb{N}, \exists n_0, \forall n \geq n_0, f(n) \leq \frac{1}{n^k}$

1.2 Higher-order CCSA logic

Squirrel uses a higher-order variant of the CCSA logic. We give a high-level overview of what the CCSA logic is and then give a formal definition of the variant.

CCSA The CCSA logic, first introduced in [4], is a first-order logic used to reason about protocols. The protocols are terms interpreted as PPTMs and the $\sim$ predicate represents computational indistinguishability. The main idea of the CCSA logic is to represent secrets, also called names, as terms in the logic. They are thus interpreted as a PPTM that will randomly sample the value. Attackers are considered as PPTMs as well and they can control the interleaving of operations.

To interpret the logic, we need a few variables:

- The random tapes ρ: every function will sample its randomness from a random tape explictly given. We usually have multiple tapes, one "honest" tape ρ_h for the secrets, and one "attacker" tape ρ_a.
- The security parameter $\eta \in \mathbb{N}$: this is used to express that probabilities are negligible for indistinguishability for example.

Note that these parameters are not included inside a model of the logic and are inputs of the semantic. For the rest of the paper, we consider η and ρ already declared.

Higher-order variant overview For greater flexibility, Squirrel relies on a higher-order variant of the CCSA logic introduced in [1]. This grants a few advantages:

- Proofs are now more reusable as they can be done in a modular way.
- Some security properties were hard to express in CCSA. For example, you can now quantify over attackers. You can now write $\forall \texttt{att} : \texttt{message} \to \texttt{message}$.

Here are the key differences:

- Terms are built with a simply-typed λ-calculus. We write $\mathcal{X}$ the set of variables.
- Terms are now interpreted as random variables over tapes.
- A *local formula* is a term of type `bool`.
- A *global formula* is a term built with a predicate, usually the $\sim$ predicate.

To differentiate between global and local logical operators, we usually use tildes for global operators. For example, the following formula is a global formula: $\tilde{\forall}(x, y).\ x \mathrel{\tilde{\Longrightarrow}} y$.

Environments and Models Let us now formally define the semantics.

A type model is a structure that provides the interpretation of the types as a set containing all possible values of the type. We write $[\![\tau]\!]^{\eta}_{\mathbb{M}}$ the interpretation of a type. Models extend type models and add interpretation of symbols. An environment $\mathcal{E}$ contains *declarations* of variables with their type, and *definitions* of variables with their type and value. We write $\mathbb{M} : \mathcal{E}$ to say that the model $\mathbb{M}$ is for the environment $\mathcal{E}$ when $\mathbb{M}$ provides an interpretation of all the variables of $\mathcal{E}$.

A model is composed of the following:

- Interpretation of symbols: they represent protocols. They are split into three parts:
  - Honest function symbols: these represent cryptographic primitives like encryption or decryption. They are deterministic.
  - Attacker function symbols: these represent attacker computations Thein interpretation use the honest tape ρ_h.
  - Name symbols: these are used to represent secrets, as they represent randomness. Their interpretation use the attacker tape ρ_a. By definition, all names are indistinguishable. Two names of the same type have the same distribution and are independent. We write $\mathcal{N} \subseteq \mathcal{X}$ the set of all names.
- An execution trace: this defines the order in which the different actors of the protocol interacted with each other. The formalism used for traces can be found in [1].

Regarding types, we might sometimes *tag* them to express some properties:

- $\texttt{finite}(\tau)$ means that $[\![\tau]\!]^{\eta}_{\mathbb{M}}$ is finite for all η.
- $\texttt{enum}(\tau)$ means that $[\![\tau]\!]^{\eta}_{\mathbb{M}}$ can be enumerated by a Turing machine in polynomial time with regard to η.

$$
\begin{aligned}
[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} &:= X_\eta(\rho) \quad \text{where } \mathbb{M}(x) = (X_\eta)_{\eta\in\mathbb{N}} && (\text{if } x \in \mathcal{E}) \\
[\![t\ t']\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} &:= [\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\![t']\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) \\
[\![\lambda(x:\tau_0).\ t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} &:= \begin{cases} [\![\tau_0]\!]^{\eta}_{\mathbb{M}} & \to [\![\tau]\!]^{\eta}_{\mathbb{M}} \\ a & \mapsto [\![t]\!]^{\eta,\rho}_{\mathbb{M}[x\mapsto \mathbb{1}^{\eta}_a]:(\mathcal{E},x:\tau_0)} \end{cases}
\end{aligned}
$$

Figure 1: Semantics of terms

Semantics A term is interpreted as a sequence of η-indexed random variables from the set of tape to the domain of the type. We write $[\![t]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ the interpretation of t with regard to η, ρ and $\mathbb{M} : \mathcal{E}$. The semantics is given in Figure 1. Note that in the definition of the semantics of the lambda term, $\mathbb{1}^{\eta}_a$ is the random variable on τ_0 such that $\mathbb{1}^{\eta}_a(\eta)(\rho) = a$ for all ρ and $\mathbb{1}^{\eta}_a(\eta')(\rho)$ is some irrelevant value when $\eta \neq \eta'$.

1.3 Computational indistinguishability in Squirrel

We can formally express the notion of indistinguishability in CCSA (and by extension the Squirrel logic) with a predicate written $\sim$. Consider two terms u and v, we say that $u \sim v$ when:

$$\forall \mathcal{A} \in \texttt{PPTM}, \eta \mapsto |\mathbb{P}_\rho(\mathcal{A}([\![u]\!]^{\eta,\rho}_{\mathbb{M}}, \rho_a) = 1) - \mathbb{P}_\rho(\mathcal{A}([\![v]\!]^{\eta,\rho}_{\mathbb{M}}, \rho_a) = 1)| \text{ is negligible in } \eta$$

1.4 Sequents

The proof system used in Squirrel is based on natural deduction. Global sequent, written $\mathcal{E}; \Theta \vdash F$, represent the global formula $\tilde{\forall}\mathcal{E}.(\tilde{\wedge}\Theta \tilde{\Longrightarrow} F)$ where:

- $\mathcal{E}$ is a set of typed variables representing an environment
- Θ is a set of global formulas used as hypothesis

2 Advanced bi-deduction

To express the composition results from [8], we leverage a technique called bi-deduction. This technique was introduced in Squirrel in [2] and is used to express cryptographic assumptions in the form of games. Intuitively, given two terms $u_0 \sim u_1$ and two cryptographic games $(\mathcal{G}_0, \mathcal{G}_1)$ that are indistinguishable, a bi-deduction expresses how there exists a simulator $\mathcal{S}$ that given u_i and $\mathcal{G}_i$ computes a term v_i. In that case, we say that (u_0, u_1) bi-deduces (v_0, v_1). Note that the simulator does not know which side (that is, which i) it is given so the same computations are used. We first start by introducing the formalism used.

2.1 Cryptographic games and oracles

Expressions To express cryptographic games, we first define a simple expression type that will interact with the game variables. We use the same formalism as in [2] that we summarize here. We consider a set of typed program variables $\mathcal{X}_p$ and a subset of the function symbols $\mathcal{L}_p \subseteq \mathcal{E}$. The expression also use a special constant b to express the side of a cryptographic game. The syntax of the expressions is described in Figure 2.

$$e ::= e_1\ e_2 \mid v \in \mathcal{X}_p \mid g \in \mathcal{L}_p \mid b$$

Figure 2: Syntax of expressions

$$\begin{array}{ll} p ::= v \leftarrow e & \mid \textbf{skip} \\ \quad \mid v \xleftarrow{\$} T[e] & \mid p_1; p_2 \\ \quad \mid v \leftarrow \mathcal{O}(\vec{e})[\vec{e_l}\ ;\ \vec{e_l}] & \mid \textbf{if } e \textbf{ then } p_1 \textbf{ else } p_2 \\ \quad \mid \textbf{abort} & \mid \textbf{while } e \textbf{ do } p \end{array}$$

Figure 3: Syntax of programs

Programs Interactions with a game are made using programs. They use a simple syntax described in Figure 3. The syntax and the semantic will be explained in greater details in Section 2.2.

Games and oracles A game is a finite set of oracles with a sequence of declarations. The declarations contain initialization of *global* variables, either through a random sampling or a particular expression. These variables are known as game variables and are different from variables in the CCSA logic. An oracle runs a simple program (a program without samplings of oracle calls) to compute an expression, after possibly sampling *local* variables. We assume that an oracle doesn't change any global variables or local variables from another oracle.

We might often talk in the rest of this paper about pairs of game that are indistinguishable. In that case, we write $\mathcal{G}$ the pair and refer to each individual game as $\mathcal{G}_i$. Similarly, we call *bi-terms* a pair of term (u_0, u_1), which we write $u_\sharp$. We access the element of side i with u_i. We also use *bi-formulas* and more generally, *bi-objects*.

Example 2.1 (PRF)**.** *The PRF cryptographic assumption is related to keyed hash functions. It states that the output of a keyed hash is indistinguishable with a fresh name. We use the same formalism as the one used in [2]. We express PRF as two indistinguishable games $\mathcal{G}_0, \mathcal{G}_1$ that are composed of two oracles:*

- *$\mathcal{O}_{hash}$: this oracle hashes its input with a key that was sampled at the beginning of the game.*
- *$\mathcal{O}_{challenge}$: this oracle behaves differently depending on the game:*
  - *In $\mathcal{G}_0$, it returns the hash of its input.*
  - *In $\mathcal{G}_1$, it returns a fresh sampling.*

To ensure there are no trivial attacks, the oracles keep track of the input that were already used and reject them if used again. A more formal definition of the oracles may be found in [2].

$$[b]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} = i$$
$$[v]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} = \mu(v) \quad \text{when } v \in \mathcal{X}_p$$
$$[g]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} = [\![g]\!]^{\eta,(\mathfrak{p}[T_A,\mathtt{bool}],\rho_0)}_{\mathbb{M}:\mathcal{E}} \quad \text{when } g \in \mathcal{X}_p$$
$$[e_1\ e_2]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu} = [e_1]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu}([e_2]^{\eta,\mathfrak{p}}_{\mathbb{M},i,\mu})$$

Figure 4: Semantics of the expressions

2.2 Programs and adversaries

Tag and sampling Now, to define adversaries, we need a clear notion of randomness' origin. Indeed, we need a way to express how an attacker can sample its own key and use it to compute a secret value. To that end, we introduce the following tags:

- T_A: represents randomness generated by the simulator $\mathcal{S}$ (i.e. the attacker) that corresponds to secrets of ρ_a.
- T_S: represents randomness of $\mathcal{S}$ that corresponds to ρ_h.
- T_G: represents randomness used by the oracles on ρ_h. The simulator cannot access these secrets.

We can tag all the randomness used by our programs (the precise syntax is given in [2]).

Oracle call When using an oracle, a program specifies the offsets on the tape where the memory is read. This allows the program to choose which term should be used. For example, the expression $v \leftarrow \mathcal{O}(x)[\mathtt{offset}_k(i); \mathtt{offset}_r(i)]$ represents the call to oracle $\mathcal{O}$. The randomness is specified in the square brackets via the `offset`() function. It takes as input a name $n \in \mathcal{N}$ of type $\tau_0 \to \tau$ and a value $a \in [\![\tau_0]\!]^{\eta}_{\mathbb{M}}$ and outputs an offset $\mathtt{offset}_n(a) \in \mathbb{N}$ which maps to the program random tape. However, the semantics of the programs have to address two issues: first, the local randomness used needs to be fresh during each call; second, the global randomness needs to be consistent across calls.

Program semantic For the semantics of these programs, memory is encoded via a *memory map* $\mu \in Mem_{\mathbb{M},\eta}$ from the variables of a program to type interpretation. Values are sampled on an infinite bitstring represented by the *program random tape* $\mathfrak{p}$. Note that there is a different bitstring for each tags and each types. The semantics of a program $(\!|p|\!)^{\eta,\mathfrak{p}}_{\mathcal{G},\mathbb{M},i,\mu}$ is the memory following the program execution against $\mathcal{G}_i$. We write $(\!|p|\!)^{\eta,\mathfrak{p}}_{\mu}$ when this is clear in the context. A variable called `res` stores the output value of a program.

Expression semantic The semantics of the expressions is given in Figure 4. Note how the semantics only use the attacker part of $\mathfrak{p}$.

Valid adversaries and secure games A program is a "valid" *adversary* when it respects the tagging and has a "correct" usage of randomness, that is fresh local randomness and consistent global randomness. An adversary also cannot access the game variables.

We define the security of a game using the classic advantage notion and quantifying over all PTIME adversaries. The interpretation of the adversaries use $\mu^{i}_{init\mathbb{M}}{}^{\eta,\mathfrak{p}}$, the *initial memory* of the game $\mathcal{G}_i$. Informally, this is defined as the memory map where all the global variable assignments are evaluated. Note that global variable samplings are evaluated during oracle calls and thus are not in the initial memory. The memory also contains the security parameter. The formal definition is given in the Appendix of [2].

2.3 Name constraints

While a program can explain where the randomness comes from using tags, we want to abstract the simulator when we are dealing with bi-deduction. However, we still need to keep in mind how to handle randomness. To that end, we introduce the notion of *name constraints*. Intuitively, each name is tagged with the following (similarly to what we have done for programs):

- T_S: indicates a name sampled by the simulator.
- $T^{loc}_{\mathcal{G}}$: indicates a name locally sampled by an oracle.
- $T^{glob}_{\mathcal{G},v}$: indicates the global sampling of the game variable v.

Note that these tags slightly differ from the ones used in programs. However, there is a clear link between both T_S, hence why we use the same symbol. In addition, $T^{loc}_{\mathcal{G}}$ and $T^{glob}_{\mathcal{G},v}$ can be seen as a more precise version of T_G.

A bit more formally, a name constraint is a tuple $c = (\vec{\alpha}, n, u, T, f)$ where $\vec{\alpha}$ is a list of variables tagged `finite`, n is a name, u is a term, T is a tag and f is a local formula. Intuitively, a name constraint c is valid when for all instantiations of the variables in $\vec{\alpha}$ where f holds, the term $(n\ u)$ is used with tag T. We then say that a constraint system C is a list of constraints. We write $C_\sharp$ a bi-constraint system.

As mentioned previously, we require the local randomness to be fresh for each call and the global randomness to be consistent. In addition to that, a name cannot be associated with two different tags. We say that a constraint system C is *valid*, written using the predicate $Valid(C)$, when it respects these three restrictions.

We also define the following shortcuts:

- $\mathcal{N}^{\eta,\rho}_{c=(\vec{\alpha},n,u,T,f),\mathbb{M}} = \{\langle n, [\![u]\!]^{\eta,\rho}_{\mathbb{M}\sigma}, T\rangle \mid \mathtt{dom}(\sigma) = \vec{\alpha}, [\![f]\!]^{\eta,\rho}_{\mathbb{M}\sigma} = \mathtt{true}\}$
- $\mathcal{N}^{\eta,\rho}_{C,\mathbb{M}} = \bigcup_{c\in C} \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}}$

These notations allow us to talk about constraints inclusion in a "correct" way.

2.4 Relation between tapes and probability coupling

So far, we have defined two types of source of randomness: program tapes $\mathfrak{p}$ for programs and random tapes ρ for terms. We need to make sure these tapes coincides on names to later mix these notions together. Thus, given a constraint system C and a model $\mathbb{M}$, we define $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ as a relation between the different tapes. We say that $\rho\ \mathcal{R}^{\eta}_{C,\mathbb{M}}\ \mathfrak{p}$ when ρ_a is a prefix of $\mathfrak{p}[T_A, \mathtt{bool}]$ and for all $(n,t,T) \in \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}}$, $\quad [\![n]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(t) = \mathfrak{p}|^{\eta}_{T}[\mathtt{offset}_n(t)]$

Note that some definitions measure probability while quantifying over ρ, while other quantify over $\mathfrak{p}$. We would like to be able to go back and forth between these probabilities. We use a *probability coupling* that we write $\mathbb{C}$. This allows us to easily switch between the marginal probabilities and the joint probabilities. Note that we want our probability coupling to respect

the relation $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ for a given C. We can build one by introducing the concept of *well-formed* constraint systems. This notion is necessary as we cannot build a coupling out of some constraint system but those are pathological cases in which we are not interested. The intuitive idea is that some names might depend on other names through the condition on the index of the constraint. As such, it is necessary to sample them in order. Well-formedness expresses that there is such an order. More details are given in [2].

2.5 Bi-deduction judgment

Memory assertions We are almost ready to define bi-deduction. The last thing we need to address is how we can keep track of the game's memory. We use pre-conditions and post-conditions to keep track of the memory evolution. We reason about memory conditions with an assertion logic. The formulas of that logic can make use of the state of the memory μ or the logical values like names using ρ. We write $\mathbb{M}, \eta, \rho, \mu \models^{A} \varphi$ the satisfaction relation.

Bi-deduction judgement Consider a program p with distinguished variables X. We say that p computes $u_\sharp \rhd_{\mathcal{G}} v_\sharp$ w.r.t. $\mathbb{M}, \eta, \rho, \mathfrak{p}, \mu, i \in \{0,1\}$ when $\mu'[\mathtt{res}] = [\![v_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ where $\mu' = (\!|p|\!)^{\eta,\mathfrak{p}}_{\mu[X \mapsto [\![u_i]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}]}$.

A bi-deduction judgement is as follows:

$$\mathcal{E}, \Theta, C_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash u_\sharp \rhd_{\mathcal{G}} v_\sharp$$

where $\mathcal{E}$ is an environment, Θ is a set of global formulas, $C_\sharp$ is a bi-constraint system, $(\varphi_\sharp, \psi_\sharp)$ are two assertion bi-formulas.

A bi-deduction judgement is *valid* when for any type structures $\mathbb{M}_0$, there exists a PTIME program p such that for all models $\mathbb{M} : \mathcal{E}$ that extends $\mathbb{M}_0$ such that $\mathbb{M} \models \Theta \tilde{\wedge} \mathit{Valid}(C_\sharp)$ holds, p is an adversary and for all security parameters $\eta \in \mathbb{N}$ and side $i \in \{0,1\}$, C_i is well-formed and for any tapes ρ $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ $\mathfrak{p}$ and for any memory μ such that $\mathbb{M}, \eta, \rho, \mu \models^{A} \varphi_i$, p computes $u_\sharp \rhd_{\mathcal{G}} v_\sharp$ w.r.t to $\mathbb{M}, \eta, \rho, \mathfrak{p}, \mu, i$ and the corresponding final memory μ' is such that $\mathbb{M}, \eta, \rho, \mu' \models^{A} \psi_i$. Lastly, we require that the computation of p relies on global sampling $G_\$$ and $L_\$$ such that:

$$G_\$ \subseteq \{\mathtt{offset}_n(t) \mid \langle n, t, T^{glob}_{\mathcal{G},v} \rangle \in \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}}, c \in C\}$$
$$L_\$ \subseteq \{\mathtt{offset}_n(t) \mid \langle n, t, T^{loc}_{\mathcal{G}} \rangle \in \mathcal{N}^{\eta,\rho}_{c,\mathbb{M}}, c \in C\}$$

Finally, we show how bi-deduction is related to computational indistinguishability.

Theorem 2.1 (Bi-Deduce)**.** *Let $\mathcal{E}$ be an environment, Θ be a set of global formulas, and $(\varphi_\sharp, \psi_\sharp)$ be bi-assertions. The following rule is sound w.r.t. models where $\mathcal{G}$ is secure and that respect the precondition:*

$$\frac{\mathcal{E}, \Theta \vdash \mathit{Valid}(C_\sharp) \qquad \mathcal{E}, \Theta, C_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash \emptyset \rhd_{\mathcal{G}} u_\sharp}{\mathcal{E}, \Theta \vdash u_0 \sim u_1}$$

Thanks to that rule, we can create a proof system to reason about bi-deduction judgements to abstract the simulator.

2.6 Example proof system

All rules for the proof system are explained in [2]. Two rules are explained here as a example.

Example 2.2 (Transitivity)**.** *The following rule expresses how we can compose simulators sequentially. Note that $C^1 \cdot C^2$ denotes the concatenation of constraints.*

$$\frac{\mathcal{E}, \Theta, C^1_\sharp, (\varphi_\sharp, \varphi'_\sharp) \vdash u_\sharp \rhd_\mathcal{G} v_\sharp \qquad \mathcal{E}, \Theta, C^2_\sharp, (\varphi'_\sharp, \psi_\sharp) \vdash u_\sharp, v_\sharp \rhd_\mathcal{G} w_\sharp}{\mathcal{E}, \Theta, C^1_\sharp \cdot C^2_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash u_\sharp \rhd_\mathcal{G} v_\sharp, w_\sharp}$$

Example 2.3 (Name)**.** *This rule here expresses how the simulator may sample its own secrets and use them. Here, $(u_\sharp | f_\sharp)$ represents the term ($f_\sharp$, if $f_\sharp$ then $u_\sharp$). We sometimes simply write $u_\sharp$ when the condition does not matter.*

$$\frac{\mathcal{E}, \Theta, C_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash u_\sharp \rhd_\mathcal{G} (v_\sharp | f_\sharp)}{\mathcal{E}, \Theta, C_\sharp \cdot \{(\emptyset, n, v_\sharp, T_S, f_\sharp)\}, (\varphi_\sharp, \psi_\sharp) \vdash u_\sharp \rhd_\mathcal{G} (n\ v_\sharp | f_\sharp)}$$

3 Shared Secrets

In this Section, we introduce the problem with shared secrets more in-depth and give a theorem to use in Squirrel. Note that the first subsections try to show the similarities between our approach and $\mathcal{O}$-indistinguishability and $\mathcal{O}$-simulatability. These notions haven't been explained in details before as they are *not* necessary to understand our final result. We kept these comparisons as it might ease the comprehension in some cases and helps in adapting other results from [8] in Squirrel.

3.1 Introducing the problem with shared secrets

Deterministic encryption example To illustrate the problem with shared secrets, consider the following example where $u = h(0, \mathsf{sk})$, $v = h(1, \mathsf{sk})$ and $w = \mathsf{sk}'$, that is, u and v are hashes of different message but with the same key and w is a key. We examine the formula $F := u \sim v \implies u, w \sim v, w$ and whether it holds or not.

With shared secrets If $\mathsf{sk} = \mathsf{sk}'$, then we can use the w (that leaks the key) to compute the hash and distinguish u and v. Thus, F does not hold when u and w share secrets.

Without shared secrets Suppose that $\mathcal{N}(u) \cap \mathcal{N}(w) = \mathcal{N}(v) \cap \mathcal{N}(w) = \emptyset$, that is $\mathsf{sk} \neq \mathsf{sk}'$. Note how we are not stating any restrictions regarding shared secrets of u and v. To prove that F holds, let us build a distinguisher $\mathcal{D}'$ for u and v using a distinguisher $\mathcal{D}$ for (u, w) and (v, w).

To use $\mathcal{D}$, we need to feed it w as input. While we do not have access to sk' as it is a secret, we can *resample* it. We write $\mathcal{S}$ the simulator that resamples w. Its output has the *same distribution* as w. Additionally, w is *independent* from u and v as they do not share any secrets. Thus, we get that $(u, \mathcal{S})$ has the same distribution as (u, w). We can use the same arguments for v.

We then define $\mathcal{D}'(x) := \mathcal{D}(x, \mathcal{S}())$. $\mathcal{D}$ runs on an input with the same distribution as (u, w) (or (v, w)) that it can distinguish. In the end, $\mathcal{D}'$ has a non-negligible advantage in distinguishing u and v.

Takeaway The point with this example is that we are *simulating* w with the simulator $\mathcal{S}$.

3.2 Encapsulating shared secrets in oracles

The example Now we want to deal with shared secrets. Let us consider the PRF example introduced in Example 2.1. Let us introduce a few terms that we will use throughout this example:

- Let $u_0 := h(\langle 1, t\rangle, \mathsf{sk})$
- Let $u_1 := n_{PRF}$ be a fresh name.
- Let $s := \mathsf{seq}_i[h(\langle 0, t_i\rangle, \mathsf{sk})]$ be a sequence of terms.

Our goal in this example is to show that: $u_0, s \sim u_1, s$. Notice how the same secret key sk is used in u_0 and s. Intuitively, in this example we show the security of the protocol u_0 *even in the presence of* s. The messages are tuples here: this usage is similar to session numbers or protocol versions. Note that this example does not use any local randomness i.e. salt in the hash, making it a "simple" example.

Proof sketch Consider the following function: $g : x \mapsto h(\langle 0, x\rangle, \mathsf{sk})$. We can see g as an oracle that we write $\mathcal{O}_g$. We want to replace any use of $h(\langle 0, \cdot\rangle, \mathsf{sk})$ by a call to g. It masks any use of the shared secret sk by g.

We can see how this example is related to composition as expressed in the composition results from [8]. We use a similar proof as what we would do to apply this theorem. This is not a formal proof per se as the formalism used is widely different. We revisit that proof later in Section 3.5 to apply new results and get a formal proof.

The oracle $\mathcal{O}_g$ computes any terms of the sequence s. This intuitively means that s is $\mathcal{O}_g$ – simulatable, i.e. can be simulated by an attacker with access to $\mathcal{O}_g$.

We then prove that $u_0 \sim_{\mathcal{O}_g} u_1$. Thanks to the PRF assumption, we already know that $u_0 \sim u_1$ as both those terms are the output of $\mathcal{O}_{challenge}$ in $\mathcal{G}_0$ and $\mathcal{G}_1$. However, g is not capable of computing u_0 or u_1 or any term related. As such, we can conclude that $u_0 \sim_{\mathcal{O}_g} u_1$.

We thus conclude *intuitively* using composition results from [8] that $u_0, s \sim u_1, s$.

Takeaway The key takeaway from this example is that to handle shared secrets, we try to encapsulate how they are used inside an oracle. We then check $\mathcal{O}$-simulatability and $\mathcal{O}$-indistinguishability with regard to that oracle.

3.3 Intuition: $\mathcal{O}$-simulatability and bi-deduction

We want to compare the following notions : $\mathcal{E}, \Theta, C_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash \emptyset \rhd_{\mathcal{O}} P$ and $\mathcal{O}$-simulatability. Note how the game in the bi-deduction judgement is the oracle $\mathcal{O}$. While our definition of a game does not match a single oracle, we implicitely use a trivial transformation. In this example, we use mono-deduction, that is, bi-deduction where both sides are the same. While these two statements use different formalism, we give an intuition on how the statements are related.

Intuitive meaning First, we can examine from a high-level perspective what these two statements entails:

- Bi-deduction means that given $\emptyset$, that is nothing, there is an adversary p that computes P while having access to the game, here $\mathcal{O}$.
- $\mathcal{O}$-simulatability means that any computation using P can use another program $\mathcal{A}^{\mathcal{O}}$ instead.

We can see that both statements suggests the existence of some "program" (either p or $\mathcal{A}^{\mathcal{O}}$) that relies on $\mathcal{O}$ and that computes P.

While it is tempting to say that both statements are equivalent, this is not easy to prove. Indeed, when looking at the definitions of each statement, we face a few obstacles.

Quantifiers The most important problem is the order of quantifiers:

- The bi-deduction judgement states that "there exists a program such that for all models of our logic…"
- $\mathcal{O}$-simulatability first specify a model and then states the existence of machine $\mathcal{A}^{\mathcal{O}}$.

This seems to be the biggest limiting factor when trying to prove an equivalence.

Formalism This difference is due to the formalism used:

- Bi-deduction uses programs whose semantics is given through a model later thanks to the symbols.
- $\mathcal{O}$-simulatability uses Turing Machines, meaning you have to give a semantic to a machine when declaring it.

While this quantifier order seems to make it impossible to prove the equivalence, we still have one "trivial" implication. We do not make a formal proof due to the different formalism used but it seems correct to say that bi-deduction implies $\mathcal{O}$-simulatability. This is why we chose this composition result to implement and bi-deduction.

3.4 Using $\mathcal{O}$-simulatability in the proof system

We want to use the composition results from [8] in Squirrel. There are some similarities between $\mathcal{O}$-simulatability and bi-deduction, as both express the existence of a simulator, as explained in Section 3.3. This is why we decided to integrate $\mathcal{O}$-simulatability to bi-deduction and its proof system.

We base our approach on the following idea. The composition results rely on two concepts: $\mathcal{O}$-simulatability and $\mathcal{O}$-indistinguishability. While $\mathcal{O}$-simulatability can be expressed using bi-deduction and a specific game, we still need to express $\mathcal{O}$-indistinguishability. To that end, we will use an astutely chosen lambda term in the Squirrel logic to represent that oracle $\mathcal{O}$. We then have an analogous concept where two terms are $\mathcal{O}$-indistinguishable when they are indistinguishable in presence of said lambda term.

Note however that we want to work with shared secrets. This means that we need to be extra careful on which secrets are used where and how. The goal of that oracle is to "hide" the shared secrets inside the lambda term. Thankfully, by adding restrictions on how to use the shared variables in the bi-deduction, we are able to ensure a correct usage of said variables.

Note that we restrict ourselves to *stateless* oracles. One of the reason is that we are not able to translate a stateful oracle to a lambda term. This is the same requirements as $\mathcal{O}$-simulatability does not work with stateless oracles.

Before formally stating the rule, we introduce a relation so the "astutely chosen lambda term" is, indeed, astutely chosen. The relation is simple: because we want the lambda to simulate the oracle, we want the lambda to output *the same terms*.

Definition 3.1. *Consider a game $\mathcal{G}$ with a single oracle $\mathcal{O}$ whose support is $\{v\}$, a name $\mathsf{sk} \in \mathcal{N}$ and an index for that name $\mathtt{t}$. We define its one-shot program evaluator:*

$$p_{\mathcal{G}}^{(\mathsf{sk}\ \mathtt{t})} := (\mathtt{res} \leftarrow \mathcal{O}(X)[\mathtt{offset}_{\mathsf{sk}}(), Y])$$

Definition 3.2. *Consider a game $\mathcal{G}$ with a single oracle $\mathcal{O}$ whose support is $\{v\}$ of type τ_{sk}, an order 1 term $\lambda_{\mathcal{G}}$ of type $\tau_1 \to \tau_2$ "compatible with $\mathcal{O}$", a name $\mathsf{sk} \in \mathcal{N}$ of type $\tau_0 \to \tau_{\mathsf{sk}}$, with $[\![\tau_0]\!]^\eta_{\mathbb{M}}$ finite for all η, and an index $\mathtt{t}$ of type τ_0. We will say that $\lambda_{\mathcal{G}} \equiv_{\mathsf{sk}\ \mathtt{t}} \mathcal{G}$ if and only if:*

For all $\eta, \mathbb{M}, \mathcal{E}, \mu, i$ with $\mathbb{M} : \mathcal{E}$, for all probability coupling $\mathbb{C}$, for all $(x : \tau_1) \in \mathcal{E}$, for all $(\vec{y} : \boldsymbol{int}) \in \mathcal{E}$ and $s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_{\mathbb{M}}$:

$$\begin{aligned}
&\forall (dc_{\rho,\mathfrak{p}})_{(\rho,\mathfrak{p})\in\mathbb{C}} \in Dist_{\rho,\mathfrak{p}}([\![\tau_2]\!]^\eta_{\mathbb{M}}),\\
&\quad \mathbb{P}_{(\rho,\mathfrak{p})\in\mathbb{C}}(\llparenthesis p_{\mathcal{G}}^{(\mathsf{sk}\ \mathtt{t})} \rrparenthesis^{\eta,\mathfrak{p}}_{\mathcal{G},\mathbb{M},i,\mu[X\mapsto[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, Y\mapsto[\![\vec{y}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}]}[\mathtt{res}] = dc_{\rho,\mathfrak{p}} \mid \mathfrak{p}|^\eta_{T_G}[\mathtt{offset}_{\mathsf{sk}}([\![\mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})] = s) =\\
&\qquad\qquad \mathbb{P}_{(\rho,\mathfrak{p})\in\mathbb{C}}([\![\lambda_{\mathcal{G}}(x,\vec{y})]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk}\ \mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)
\end{aligned}$$

Note that $\mathcal{O}$ may locally sample values, for example for non-deterministic encryption. This is why we allow $\lambda_{\mathcal{G}}$ to have other names inside. We can only require the distribution equality, as it would be infeasible otherwise.

Based on this generic definition, we show that using a coupling contained in some $\mathcal{R}^\eta_{C,\mathbb{M}}$ allows us to rewrite this definition in a slightly nicer style. This let us to work with the same coupling that the bi-deduction gives us.

Lemma 3.1. *Consider a game $\mathcal{G}$ with a single oracle $\mathcal{O}$ whose support is $\{v\}$ of type τ_{sk}, an order 1 term $\lambda_{\mathcal{G}}$ of type $\tau_1 \to \tau_2$ "compatible with $\mathcal{O}$", a name $\mathsf{sk} \in \mathcal{N}$ of type $\tau_0 \to \tau_{\mathsf{sk}}$ and an index $\mathtt{t}$ of type τ_0.*

If $\lambda_{\mathcal{G}} \equiv_{\mathsf{sk}\ \mathtt{t}} \mathcal{G}$, then for all $\eta, \mathbb{M} : \mathcal{E}, C, \mu, i$ and any C such that given $C' := C \cdot \{(\emptyset, \mathsf{sk}, \mathtt{t}, T^{glob}_{\mathcal{G},v}, \top)\}$, C' is well-formed and valid, for all probability coupling $\mathbb{C}$ contained in $\mathcal{R}^\eta_{C',\mathbb{M}}$, for any variable $(x : \tau_1) \in \mathcal{E}$ that is computable in polynomial time, for all $(\vec{y} : \boldsymbol{int}) \in \mathcal{E}$ and for any $s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_{\mathbb{M}}$:

$$\begin{aligned}
&\forall (dc_{\rho,\mathfrak{p}})_{(\rho,\mathfrak{p})\in\mathbb{C}} \in Dist([\![\tau_2]\!]^\eta_{\mathbb{M}}),\\
&\qquad \mathbb{P}_{(\rho,\mathfrak{p})\in\mathbb{C}}(\llparenthesis p_{\mathcal{G}}^{(\mathsf{sk}\ \mathtt{t})} \rrparenthesis^{\eta,\mathfrak{p}}_{\mathcal{G},\mathbb{M},i,\mu[X\mapsto[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, Y\mapsto[\![\vec{y}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}]}[\mathtt{res}] = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk}\ \mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) =\\
&\qquad\qquad \mathbb{P}_{(\rho,\mathfrak{p})\in\mathbb{C}}([\![\lambda_{\mathcal{G}}(x,\vec{y})]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk}\ \mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)
\end{aligned}$$

Proof. The proof here is straight-forward. Because $\lambda_{\mathcal{G}} \equiv_{\mathsf{sk}\ \mathtt{t}} \mathcal{G}$, we only need to show that $\mathfrak{p}|^\eta_{T_G}[\mathtt{offset}_{\mathsf{sk}}([\![\mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})] = [\![\mathsf{sk}\ \mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$. This is the case because the coupling we use is contained in $\mathcal{R}^\eta_{C',\mathbb{M}}$. □

Note how we set the global variables of the game implicitly in the relation: this will be used to represent shared secrets. Additionally, the values on both sides of the game have the same value. However, because $\lambda_{\mathcal{G}}$ is not able to infer which side it is on, there shouldn't be any conversion possible when an oracle uses sides. This should not be an issue given that our goal is to model oracles that are doing cryptographic operations and not games per se.

The rule we introduce is based on the Bi-Deduce rule:

Theorem 3.1 (Composition Bi-Deduce)**.** *Let $\mathcal{G}$ be a game with only one (stateless) oracle $\mathcal{O}$ and v be its support of type τ_{sk}. Let $\mathcal{E}$ be an environment, Θ be a set of global formulas, $C_\sharp$ be a well-formed constraint system and $(\varphi_\sharp, \psi_\sharp)$ be assertion bi-formulas such that for all $i \in \{0,1\}, \eta, \rho\ \mathcal{R}^\eta_{C,\mathbb{M}}\ \mathfrak{p}$, $\mathbb{M}, \eta, \rho, \mu^i_{init\mathbb{M}}{}^{\eta,\mathfrak{p}} \models^A \varphi_i$. Let u, v be terms. Let w be an order one term of*

type $\tau_1 \to \tau_2$. *Let* sk *be a name of type* $\tau_0 \to \tau_{\mathsf{sk}}$ *and an index* $\mathtt{t}$ *of type* τ_0. *The following rule is sound:*

$$\textsf{CBD}\frac{\mathcal{E}, \Theta \vdash Valid(C'_\sharp) \quad \mathcal{E}, \Theta, C'_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash \emptyset \rhd_{\mathcal{G}} w \quad \mathcal{E}, \Theta \vdash u, \lambda_{\mathcal{G}} \sim v, \lambda_{\mathcal{G}} \quad \mathcal{N}(w) \cap \mathcal{N}(u, v) = \{\mathsf{sk}\ \mathtt{t}\}}{\mathcal{E}, \Theta \vdash u, w(u) \sim v, w(v)}$$

where:

- $C'_\sharp = C_\sharp \cdot \{(\emptyset, \mathsf{sk}, \mathtt{t}, T^{glob}_{\mathcal{G},v}, \top)\}$
- $\lambda_{\mathcal{G}} \equiv_{\mathsf{sk}\ \mathtt{t}} \mathcal{G}$
- *Due to the current restrictions on first-order bi-deduction, we can only bi-deduction judgements when* $\mathtt{enum}(\tau_1)$ *holds. This may be lifted in the future.*

Here are a few things to note regarding this rule. First, we are only dealing with a single oracle and a single shared secret. This rule can be expanded to support multiple shared secrets and oracles, for example by iterating or adapting the proof. Second, the game has a single global variable. This makes the proof slightly easier, while not being too restrictive. This helps keeping the oracle used "minimal".

Proof. The idea of the proof is straight-forward: we consider a simulator for w given by the bi-deduction judgement. We transform it so it uses $\lambda_{\mathcal{G}}$ instead of $\mathcal{O}$. We then use it in a distinguisher of $u, w(u)$ and $v, w(v)$ to create a distinguisher of $u, \lambda_{\mathcal{G}}$ and $v, \lambda_{\mathcal{G}}$.

Now formally. Let p be a program used to bi-deduce w. Let $\mathcal{D}$ be a distinguisher of $u, w(u)$ and $v, w(v)$. We now want to build a distinguisher $\mathcal{D}\,'$ of $u, \lambda_{\mathcal{G}}$ and $v, \lambda_{\mathcal{G}}$.

Program Transformation First, we need to transform p to a PPTM, so it can be used inside a distinguisher. There are a few points here that we need to be careful about. We create a PPTM $\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}$ that takes an oracle $\lambda_{\mathcal{G}}$ as input. Additionally, $\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}$ is also given access to η (as unary) and ρ_a, similarly to what distinguishers are supplied with. This PPTM will do the same computations as p. $\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}$ returns $\mathtt{res}$ from the memory.

When evaluating a program, we use a semantic that depends on a starting memory μ. $\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}$ implicitely use the initial memory of the game $\mathcal{G}$, $\mu^{i\ \eta,\mathfrak{p}}_{init\mathbb{M}}$. Given how simple the game is, this initial memory only has η inside. In particular, it is independent of $\mathfrak{p}$, i and $\mathbb{M}$. We thus refer to that memory as μ_{init}. Note that we will later need to use a different starting memory for the correction proof. Thus, we write $\widetilde{\mathcal{P}^{\mu}_{\lambda_{\mathcal{G}}}}$ to specify the starting memory. We also write $\mu{-}\mathsf{tape}(\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}})$ the memory after the execution $\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}$.

There are two operations a program can do that cannot be simulated on a valid adversary PPTM:

- *Oracle Calls.* This is where $\lambda_{\mathcal{G}}$ will be used. The idea is to replace every oracle call by a call to $\lambda_{\mathcal{G}}$. We do have to handle global and local variable offsets that are supplied during a program oracle call. For global offsets, thanks to our restrictions regarding the support of the oracle, we know that it has to be $\mathsf{sk}\ \mathtt{t}$: we can ignore this as a valid $\lambda_{\mathcal{G}}$ should handle it. For local offsets, we expect $\lambda_{\mathcal{G}}$ to take these offsets as input and correctly relay the arguments.

  To sum it up, we have the following transformation:

$$v \leftarrow \mathcal{O}(\vec{e})[\mathtt{offset}_{\mathsf{sk}}([\![\mathtt{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}); \vec{e_l}]$$
$$\downarrow$$
$$v \leftarrow [\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\vec{e}]^{\eta,\mathfrak{p}}_{\mu}, [\vec{e_l}]^{\eta,\mathfrak{p}}_{\mu})$$

- *Random Samplings.* While a PPTM can do random sampling, it can only do so on the tape ρ_a. A program, on the other hand, can sample on ρ_h using the tag T_S. We resample all T_S randomness on ρ_a, using a disjoint part of it, so we make sure there is no overlap with other uses. Randomness tagged with T_A is kept the same as it uses ρ_a already. Randomness tagged with T_G cannot happen as p is a valid adversary.
- *Expressions.* Remember how expressions rely only on the attacking part of $\mathfrak{p}$. Due to the relation $\mathcal{R}^{\eta}_{C,\mathbb{M}}$ between tapes, that means that expressions depend on ρ_a. They are thus computable by the PPTM $\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}$. As such, we might write $[e]^{\eta,\rho_a}_{\mu}$ to express that we compute them.

Finally, we consider $\mathcal{P}_{\lambda_\mathcal{G}}$ that instead computes $(x, p(x))$, that is $\mathcal{P}_{\lambda_\mathcal{G}}(x, \lambda_\mathcal{G}) := (x, \widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}(\lambda_\mathcal{G})(x))$ returning its argument, so it can be composed with $\mathcal{D}$ later. Importantly, the construction of $\mathcal{P}_{\lambda_\mathcal{G}}$ and $\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}$ does not depend on the security parameter η.

Correction proof Let η be the security parameter that will be used throughout the rest of the proof. Before we begin our probability computations, note that because C is well-formed, C' is well-formed as well. To justify that, remember how well-formedness intuitively expresses the need of an ordering when sampling variables. The condition is $\top$ so it doesn't add any complexity and t might appear in C but because it is already well-formed, it doesn't create a problem. Thus, thanks to a lemma from [8], we know that we have a probability coupling contained in $\mathcal{R}^{\eta}_{C',\mathbb{M}}$. We use that coupling implicitly for the rest of the proof. That is, all probabilities are measured over ρ and / or $\mathfrak{p}$ and the coupling allows us to switch at will.

Now that we have built $\mathcal{P}_{\lambda_\mathcal{G}}$, we want to use it inside the distinguisher $\mathcal{D}$. More precisely, we need to show that $\mathcal{P}_{\lambda_\mathcal{G}}$ has the same distribution as $(x, w(x))$ for each values of $\mathsf{sk\ t}$. Formally, we want to prove the following:

$$\begin{aligned}\forall (x:\tau_1) \in \mathcal{E},\ c \in [\![\tau_1 \times \tau_2]\!]^{\eta}_{\mathbb{M}},\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\\ \mathcal{N}(w)\ \cap\ \mathcal{N}(x) = \{\mathsf{sk\ t}\} \implies\\ \mathbb{P}([\![x, w(x)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) = \mathbb{P}(\mathcal{P}_{\lambda_\mathcal{G}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \quad (1)\end{aligned}$$

Note that we are not simply requiring the same distributions: we need the same distribution for all values of the secret key. This ensures that both the PPTM and the terms use the same value for $\mathsf{sk\ t}$ and are not simply resampling it. Furthermore, we require that the argument x shares $\mathsf{sk\ t}$ (and only $\mathsf{sk\ t}$) with w.

Transformation correction Let us first prove that the transformation of p is correct. To that end, we prove:

$$\begin{aligned}\forall a \in [\![\tau_1]\!]^{\eta}_{\mathbb{M}},\ c \in [\![\tau_2]\!]^{\eta}_{\mathbb{M}},\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\\ \mathbb{P}((\!|p|\!)^{\eta,\mathfrak{p}}_{\mu_{init}}[\mathtt{res}](a) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) = \mathbb{P}(\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})(a) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \quad (2)\end{aligned}$$

We prove by induction on programs that the memory after the execution of a program has the same distribution whether we use the program or its translation to a PPTM. We use the following induction hypothesis on p:

$$\begin{aligned}\forall d\mu^i_{\rho,\mathfrak{p}} \in Dist(Mem_{\mathbb{M},\eta}),\ D\mu^f_{\rho,\mathfrak{p}} \subseteq Dist(Mem_{\mathbb{M},\eta}),\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\\ \mathbb{P}((\!|p|\!)^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}} \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) = \mathbb{P}(\mu\text{−}\mathsf{tape}(\widetilde{\mathcal{P}^{d\mu^i_{\rho,\mathfrak{p}}}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})) \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)\\ (\mathcal{H}(p))\end{aligned}$$

Note that this induction property does imply (2).

Now for the proof. Here are the key points:

- *Oracle calls*: consider the following program $p := (v \leftarrow \mathcal{O}(e)[\mathtt{offset}_{\mathsf{sk}}([\![\mathsf{t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}); \vec{e_l}])$. Its transformation is $\widetilde{\mathcal{P}^{\mu}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) := (v \leftarrow ([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([e]^{\eta,\rho_a}_{\mu}, [\vec{e_l}]^{\eta,\rho_a}_{\mu})))$. To show the induction property, we only have to show that v is the same as there are no side effects due to the oracle being stateless. Formally, we have to show the following:

$$\forall d\mu^i_{\rho,\mathfrak{p}} \in Dist(Mem_{\mathbb{M},\eta}),\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\ dc_{\rho,\mathfrak{p}} \in Dist([\![\tau_2]\!]^{\eta}_{\mathbb{M}}),$$
$$\mathbb{P}(\langle\!|p|\!\rangle^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}}[v] = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) =$$
$$\mathbb{P}(\mu\text{−}\mathsf{tape}(\widetilde{\mathcal{P}^{d\mu^i_{\rho,\mathfrak{p}}}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}))[v] = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \quad (3)$$

First, given the definition of $p^{(\mathsf{sk\ t})}_{\mathcal{G}}$, the following holds:

$$\forall \mu \in Mem_{\mathbb{M},\eta},\ \forall \mathfrak{p}, \quad \langle\!|p|\!\rangle^{\eta,\mathfrak{p}}_{\mu}[v] = \langle\!|p^{(\mathsf{sk\ t})}_{\mathcal{G}}|\!\rangle^{\eta,\mathfrak{p}}_{\mu[X\mapsto[e]^{\eta,\rho_a}_{\mu}, Y\mapsto[\vec{e_l}]^{\eta,\rho_a}_{\mu}]}[\mathtt{res}] \quad (4)$$

We can now prove (3):

$$\begin{aligned}
&\forall d\mu^i_{\rho,\mathfrak{p}} \in Dist(Mem_{\mathbb{M},\eta}),\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\ dc_{\rho,\mathfrak{p}} \in Dist([\![\tau_2]\!]^{\eta}_{\mathbb{M}}),\\
&\mathbb{P}(\langle\!|p|\!\rangle^{\eta,\mathfrak{p}}_{\mu}[v] = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)\\
&= \mathbb{P}(\langle\!|p^{(\mathsf{sk\ t})}_{\mathcal{G}}|\!\rangle^{\eta,\mathfrak{p}}_{\mu[X\mapsto[e]^{\eta,\rho_a}_{\mu}, Y\mapsto[\vec{e_l}]^{\eta,\rho_a}_{\mu}]}[\mathtt{res}] = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (4)\\
&= \mathbb{P}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([e]^{\eta,\rho_a}_{\mu}, [\vec{e_l}]^{\eta,\rho_a}_{\mu}) = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (\text{Lemma 3.1})\\
&= \mathbb{P}(\mu\text{−}\mathsf{tape}(\widetilde{\mathcal{P}^{\mu}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}))[v] = dc_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}\ \text{definition})
\end{aligned}$$

Note that while Lemma 3.1 uses variables in the environment and not expressions, we can still use it here as quantifying over variables is the same as quantifying over random variables.

- *Samplings*: A sampling on the tape is kept the same in the transformation. As explained in the transformation, we are not able to sample when the value is tagged T_S. When resampling on ρ_a, we use a different part of the tape ρ_a. This means that the new samplings are still independent of each others. Thus, we keep the same distribution.

- For the rest of the cases, we only focus on $(p_1; p_2)$ as an example. This case justifies why we deal with memory in the induction property. We write $\widetilde{\mathcal{P}_{\lambda_\mathcal{G},i}}$ the transformation of p_i and $\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}$ the transformation of $(p_1; p_2)$.

  We have the following equations:

$$
\begin{aligned}
&\forall d\mu^i_{\rho,\mathfrak{p}} \in Dist(Mem_{\mathbb{M},\eta}),\ D\mu^f_{\rho,\mathfrak{p}} \subseteq Dist(Mem_{\mathbb{M},\eta}),\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\\
&\mathbb{P}(\langle\!|p_1;p_2|\!\rangle^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}} \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)\\
&= \mathbb{P}(\langle\!|p_2|\!\rangle^{\eta,\mathfrak{p}}_{\langle\!|p_1|\!\rangle^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}}} \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(Definition of ;)}\\
&= \mathbb{P}(\mathcal{P}^{\widetilde{\langle\!|p_1|\!\rangle^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}}}}_{\lambda_{\mathcal{G}},2} \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(cf 1.)}\\
&= \mathbb{P}(\langle\!|p_1|\!\rangle^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}} \in (\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}},2}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}))^{-1}(D\mu^f_{\rho,\mathfrak{p}}) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(cf 2.)}\\
&= \mathbb{P}(\widetilde{\mathcal{P}^{d\mu^i_{\rho,\mathfrak{p}}}_{\lambda_{\mathcal{G}},1}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) \in (\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}},2}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}))^{-1}(D\mu^f_{\rho,\mathfrak{p}}) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(cf 3.)}\\
&= \mathbb{P}(\mathcal{P}^{\widetilde{\mathcal{P}^{d\mu^i_{\rho,\mathfrak{p}}}_{\lambda_{\mathcal{G}},1}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})}}_{\lambda_{\mathcal{G}},2}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(cf 4.)}\\
&= \mathbb{P}(\widetilde{\mathcal{P}^{d\mu^i_{\rho,\mathfrak{p}}}_{\lambda_{\mathcal{G}}}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) \in D\mu^f_{\rho,\mathfrak{p}} \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(Definition of } \widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}})
\end{aligned}
$$

where:

1. We use $\mathcal{H}(p)$ on p_2 with the memory distributions $\langle\!|p_1|\!\rangle^{\eta,\mathfrak{p}}_{d\mu^i_{\rho,\mathfrak{p}}}$ and $D\mu^f_{\rho,\mathfrak{p}}$.
2. We use the inverse image of the function $(\mu \mapsto \widetilde{\mathcal{P}^{\mu}_{\lambda_{\mathcal{G}},2}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}))$
3. We use $\mathcal{H}(p)$ on p_1 with the memory distributions $d\mu^i_{\rho,\mathfrak{p}}$ and $(\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}},2}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}))^{-1}(D\mu^f_{\rho,\mathfrak{p}})$
4. We reverse step 2 using the inverse image again.

This concludes this case.

Lifting to $(x, w(x))$ Note that p computes w with the bi-deduction judgement, thus we have that:

$$\forall \rho\ \mathcal{R}^{\eta}_{C,\mathbb{M}}\ \mathfrak{p}, \quad \langle\!|p|\!\rangle^{\eta,\mathfrak{p}}_{\mu_{init}}[\mathtt{res}] = [\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} \tag{5}$$

Replacing inside Equation 2, we get:

$$\forall a \in [\![\tau_1]\!]^{\eta}_{\mathbb{M}},\ c \in [\![\tau_2]\!]^{\eta}_{\mathbb{M}},\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},$$
$$\mathbb{P}([\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(a) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) = \mathbb{P}(\widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})(a) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \tag{6}$$

However, we want to "lift" that result to $(x, w(x))$ so we that we can use it with the distinguisher $\mathcal{D}$. To do that, we will need to use the hypothesis regarding shared names. Let us show that the following equation holds:

$$\begin{aligned}
&\forall (x : \tau_1) \in \mathcal{E},\ s \in [\![\tau_{\mathsf{sk}}]\!]^{\eta}_{\mathbb{M}},\ c \in [\![\tau_1 \times \tau_2]\!]^{\eta}_{\mathbb{M}},\\
&\qquad \mathcal{N}(w) \cap \mathcal{N}(x) = \{\mathsf{sk\ t}\} \implies\\
&\qquad \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, [\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) =\\
&\qquad\qquad \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, \widetilde{\mathcal{P}_{\lambda_{\mathcal{G}}}}([\![\lambda_{\mathcal{G}}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)
\end{aligned} \tag{7}$$

We are trying to add $[\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}$ on both sides. The justification is as follows. We know that x and w only share $\mathsf{sk\ t}$. However, we are working with conditional probabilities where $\mathsf{sk\ t}$ is

fixed. Thus, x and w (when seen as random variables over tapes) are conditionally independent. For similar reasons, x and $\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}(\cdot, \lambda_\mathcal{G})$ are also conditionally independent. Note that the two independence we just mentioned concern w and *not* $w(x)$. However, the following equivalence is true by definition of the semantic:

$$\forall \rho,\ (x : \tau_1) \in \mathcal{E},\ (c_1, c_2) \in [\![\tau_1 \times \tau_2]\!]^\eta_\mathbb{M}, \quad [\![x, w(x)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = (c_1, c_2) \iff [\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c_1 \wedge [\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(c_1) = c_2$$

Then, using probabilities on both sides, we get the following:

$$\begin{aligned}\forall s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_\mathbb{M},\ (x : \tau_1) \in \mathcal{E},\ (c_1, c_2) \in [\![\tau_1 \times \tau_2]\!]^\eta_\mathbb{M},& \\ \mathbb{P}([\![x, w(x)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = (c_1, c_2) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) =& \\ \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c_1 \wedge [\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(c_1) = c_2 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)&\end{aligned}$$

Notice how the right probability now deals with $w(c_1)$ instead of $w(x)$. This means we can use the independence between x and w. Thus, the following equation holds:

$$\begin{aligned}\forall s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_\mathbb{M},\ (x : \tau_1) \in \mathcal{E},\ (c_1, c_2) \in [\![\tau_1 \times \tau_2]\!]^\eta_\mathbb{M},& \\ \mathcal{N}(w) \cap \mathcal{N}(x) = \{\mathsf{sk\ t}\} \implies& \\ \mathbb{P}([\![x, w(x)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = (c_1, c_2) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) =& \\ \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c_1 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \cdot \mathbb{P}([\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(c_1) = c_2 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)& \quad (8)\end{aligned}$$

Using a similar trick with $\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}$, we get:

$$\begin{aligned}\forall s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_\mathbb{M},\ (x : \tau_1) \in \mathcal{E},\ (c_1, c_2) \in [\![\tau_1 \times \tau_2]\!]^\eta_\mathbb{M},& \\ \mathcal{N}(w) \cap \mathcal{N}(x) = \{\mathsf{sk\ t}\} \implies& \\ \mathbb{P}(([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, \widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, [\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})) = (c_1, c_2) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) =& \\ \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c_1 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \cdot \mathbb{P}(\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})(c_1) = c_2 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s)& \quad (9)\end{aligned}$$

Finally, we can show (7). Consider $(x : \tau_1) \in \mathcal{E}$ such that $\mathcal{N}(w) \cap \mathcal{N}(x) = \{\mathsf{sk\ t}\}$, $s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_\mathbb{M}$ and $(c_1, c_2) \in [\![\tau_1 \times \tau_2]\!]^\eta_\mathbb{M}$:

$$\begin{aligned}&\mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, [\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = (c_1, c_2) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \\ &= \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c_1 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \cdot \mathbb{P}([\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}(c_1) = c_2 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (8) \\ &= \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c_1 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \cdot \mathbb{P}(\widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})(c_1) = c_2 \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (6) \\ &= \mathbb{P}(([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, \widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})) = (c_1, c_2) \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (9)\end{aligned}$$

Finishing the proof Now, to sum it up, we can prove (1). Let $(x : \tau_1) \in \mathcal{E}$ such that $\mathcal{N}(w) \cap \mathcal{N}(x) = \{\mathsf{sk\ t}\}$, let $c \in [\![\tau_1 \times \tau_2]\!]^\eta_\mathbb{M}$ and let $s \in [\![\tau_{\mathsf{sk}}]\!]^\eta_\mathbb{M}$

$$\begin{aligned}&\mathbb{P}(\mathcal{P}_{\lambda_\mathcal{G}}([\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) \\ &= \mathbb{P}(([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, \widetilde{\mathcal{P}_{\lambda_\mathcal{G}}}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, [\![\lambda_\mathcal{G}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(by definition of } \mathcal{P}_{\lambda_\mathcal{G}}) \\ &= \mathbb{P}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}, [\![w]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}([\![x]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}}) = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && (7) \\ &= \mathbb{P}([\![x, w(x)]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = c \mid [\![\mathsf{sk\ t}]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}} = s) && \text{(by definition of } [\![\cdot]\!]^{\eta,\rho}_{\mathbb{M}:\mathcal{E}})\end{aligned}$$

We can now create the distinguisher $\mathcal{D}' := \mathcal{D} \circ \mathcal{P}_{\lambda_\mathcal{G}}$. Thanks to (1), $\mathcal{P}_{\lambda_\mathcal{G}}$ and $(x, w(x))$ have the same distribution and $\mathcal{D}$ has a non-negligible advantage against it. Thus, $\mathcal{D}'$ has a non-negligible advantage as well. Using $x = u$ or $x = v$, we created a distinguisher of $(u, \lambda_\mathcal{G})$ and $(v, \lambda_\mathcal{G})$, which concludes the proof.

□

$$\mathsf{CBD}\,\frac{\mathcal{E},\Theta \vdash Valid(C_1) \qquad \dfrac{\ldots\tau_1\ldots}{\mathcal{E},\Theta,C_1,(\top,\top)\vdash \emptyset \rhd_{\mathcal{G}} s} \qquad \mathsf{BD}\,\dfrac{\mathcal{E},\Theta \vdash Valid(C_1') \qquad \dfrac{\ldots\tau_2\ldots}{\mathcal{E},\Theta,C_1',(\varphi_0,\varphi_2)\vdash \emptyset \rhd_{PRF} (u_\sharp,\lambda_{\mathcal{G}})}}{\mathcal{E},\Theta \vdash (u_0,\lambda_{\mathcal{G}}) \sim (u_1,\lambda_{\mathcal{G}})}}{u_0, s \sim u_1, s}$$

$$\varphi_0 = \{l_{hash} \mapsto []; l_{challenge} \mapsto []\} \qquad C_1 = \{(\emptyset, \mathsf{sk}, (), T^{glob}_{\mathcal{G},v}, \top)\}$$
$$\varphi_2 = \{l_{hash} \mapsto [x]; l_{challenge} \mapsto [\langle 1, t\rangle]\} \qquad C_1' = \{(\emptyset, \mathsf{sk}, (), T^{glob}_{PRF,v}, \top)\}$$

Figure 5: Beginning of the derivation using the augmented proof system.

3.5 Example: PRF and the proof system

We now give an example of the application of Composition Bi-Deduce. We reuse the example from Section 3.2 and make a formal proof using the proof system. The start of this proof is summarized in Figure 5.

We start by applying Composition Bi-Deduce with the oracle g. More formally, we define the oracle as follows: $\mathcal{O}(x) := \texttt{return}\ h(\langle 0, x\rangle, v)$ where v is a global variable of the game $\mathcal{G}$ that we use in the rule. First, the support of $\mathcal{O}$ is v. Next, we use the following lambda: $\lambda_{\mathcal{G}} := \lambda x.h(\langle 0, x\rangle, \mathsf{sk})$. We now have to verify that $\lambda_{\mathcal{G}} \equiv_{\mathsf{sk}} \mathcal{G}$. This is immediate with our current definitions.

We can now apply the Composition Bi-Deduce rule. We prove the other indistinguishability using the classic Bi-Deduce rule.

Note that we have already instantiated constraints and memory conditions. When writing the proof by hand, you usually find the correct instantiations at the end. Note that the constraints are indeed valid.

We separate the rest of the proof in two subtrees, namely τ_1 and τ_2. We only give the idea here. The full proofs can be found in Appendix A.

- Consider the goal that τ_1 has to prove. This is done by chaining calls to the new oracle in the game to compute all terms in s. We use the Transitivity rule to chain the calls. Note that this technique creates a derivation tree roughly the same size as the sequence.
- Now for τ_2. Using Transitivity, we can first simulate $u_\sharp$ and then the oracle term itself. Simulating $u_\sharp$ is done using the challenge oracle from the PRF game. Note that this oracle creates pre-conditions and post-conditions, as it is stateful. To simulate the oracle term, we use the Lambda rule first to discharge the parameter. The proof is finished using the hashing oracle from PRF.

3.6 Implementation in Squirrel

We unfortunately did not have time to implement this result inside Squirrel. However, in this Section, we give a few pointers on how it could possibly be integrated in the tool.

There are two already existing tactics of interest:

- `crypto` $\mathcal{G}$ `(key:sk)`: this tactic closes the goal $u_\sharp$ if $\emptyset \rhd_{\mathcal{G}} u_\sharp$ and `key` is tagged $T^{glob}_{\mathcal{G},v}$.
- `deduce` w: this tactic allows one to transform a goal $u, w \sim v, w$ into $u \sim v$ when w can be computed from the rest of the terms using only function applications (i.e. no names !).

If we consider a simpler case where w is an order 0 term, that is when the rule is:

$$\frac{\mathcal{E}, \Theta \vdash Valid(C'_\sharp) \quad \mathcal{E}, \Theta, C'_\sharp, (\varphi_\sharp, \psi_\sharp) \vdash \emptyset \rhd_\mathcal{G} w \quad \mathcal{E}, \Theta \vdash u, \lambda_\mathcal{G} \sim v, \lambda_\mathcal{G} \quad \mathcal{N}(w) \cap \mathcal{N}(u, v) = \{\mathsf{sk\ t}\}}{\mathcal{E}, \Theta \vdash u, w \sim v, w}$$

Our tactic could work as follow:

1. The user supplies w, $\mathsf{sk\ t}$ and $\lambda_\mathcal{G}$.
2. The tactic checks the condition regarding shared names.
3. It generates the game $\mathcal{G}$ from $\lambda_\mathcal{G}$.
4. Use the tactic `crypto` on the game $\mathcal{G}$ with $\mathsf{sk\ t}$ to close the bi-deduction judgement.
5. Change the goal to $u, \lambda_\mathcal{G} \sim v, \lambda_\mathcal{G}$.

The generation of the game could use a simple transformation from lambda term to program. We only need to prove that the transformation yields games such that $\lambda_\mathcal{G} \equiv_{\mathsf{sk\ t}} \mathcal{G}$.

Conclusion and Future Works

In this paper, we proposed an extension of the current bi-deduction proof system of Squirrel. This extension allows us to leverage the power of composition even in presence of shared secrets. We gave an example on how it could be used in Section 3.5.

While we did not have time to implement it in the tool, it should not represent a huge hurdle as it should fit nicely with the rest of the proof system.

This result is based only on the first result from [8]. A lot of corollaries and other results are established in that paper. In the future, we want to have those other results available in Squirrel without further expanding the theory. Instead, we would be relying on this result and expressing the rest as corollaries.

References

[1] David Baelde, Adrien Koutsos, and Joseph Lallemand. "A Higher-Order Indistinguishability Logic for Cryptographic Reasoning". In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175781.

[2] David Baelde, Adrien Koutsos, and Justine Sauvage. "Foundations for Cryptographic Reductions in CCSA Logics". working paper or preprint. Aug. 2024. URL: https://hal.science/hal-04511718.

[3] David Baelde et al. "The Squirrel Prover and its Logic". In: *ACM SIGLOG News* 11.2 (2024), pp. 62–83. DOI: 10.1145/3665453.3665461. URL: https://doi.org/10.1145/3665453.3665461.

[4] Gergei Bana and Hubert Comon-Lundh. "A Computationally Complete Symbolic Attacker for Equivalence Properties". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 609–620. ISBN: 9781450329576. DOI: 10.1145/2660267.2660276. URL: https://doi.org/10.1145/2660267.2660276.

[5] Gilles Barthe et al. "Computer-Aided Security Proofs for the Working Cryptographer". In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 71–90. ISBN: 978-3-642-22792-9.

[6] Bruno Blanchet. "Composition theorems for CryptoVerif and application to TLS 1.3". In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE. 2018, pp. 16–30.

[7] Ran Canetti, Alley Stoughton, and Mayank Varia. "Easyuc: Using easycrypt to mechanize proofs of universally composable security". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE. 2019, pp. 167–16716.

[8] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. "Oracle Simulation: A Technique for Protocol Composition with Long Term Shared Secrets". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by Jay Ligatti et al. ACM, 2020, pp. 1427–1444. DOI: 10.1145/3372297.3417229. URL: https://doi.org/10.1145/3372297.3417229.

[9] François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. "Bringing state-separating proofs to EasyCrypt a security proof for cryptobox". In: *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*. IEEE. 2022, pp. 227–242.

A PRF Example

$$\text{TR}\;\frac{\mathcal{O}\;\dfrac{\mathcal{E},\Theta,C_0,(\psi,\psi)\vdash\emptyset\rhd_{\mathcal{G}}\langle 0,t_0\rangle \quad \Theta\models\{\psi\}h(\langle 0,t_0\rangle,\mathsf{sk})\leftarrow\mathcal{O}_g(\langle 0,t_0\rangle)[\mathsf{sk};.]\{\psi\}}{\mathcal{E},\Theta,C_1,(\psi,\psi)\vdash\emptyset\rhd_{\mathcal{G}}h(\langle 0,t_0\rangle,\mathsf{sk})} \qquad \text{TR}\;\dfrac{\ldots}{\mathcal{E},\Theta,C_1,(\psi,\psi)\vdash\emptyset\rhd_{\mathcal{G}}\mathsf{seq}_{i\geq 1}[h(\langle 0,t_i\rangle,\mathsf{sk})]}}{\mathcal{E},\Theta,C_1,(\psi,\psi)\vdash\emptyset\rhd_{\mathcal{G}}s}$$

with:

$$\psi=\top$$

$$C_0=\{\}$$
$$C_1=\{(\emptyset,\mathsf{sk},(),T^{glob}_{\mathcal{G},v},\top)\}$$
Note that $C_1\cdot C_1=C_1$

Figure 6: Derivation of τ_1

$$\text{TR}\;\frac{\text{Lambda}\;\dfrac{\mathcal{O}_{hash}\;\dfrac{(\mathcal{E},x),\Theta,C_0',(\varphi_0,\varphi_0)\vdash x\rhd_{PRF}\langle 0,x\rangle \quad \Theta\models\{\varphi_0\}h(\langle 0,x\rangle,\mathsf{sk})\leftarrow\mathcal{O}_{hash}(\langle 0,x\rangle)[\mathsf{sk};.]\{\varphi_1\}}{(\mathcal{E},x),\Theta,C_1',(\varphi_0,\varphi_1)\vdash x\rhd_{PRF}h(\langle 0,x\rangle,\mathsf{sk})}}{\mathcal{E},\Theta,C_1',(\varphi_0,\varphi_1)\vdash\emptyset\rhd_{PRF}\lambda_{\mathcal{G}}} \qquad \mathcal{O}_{challenge}\;\dfrac{(\mathcal{E},x),\Theta,C_0',(\varphi_1,\varphi_1)\vdash\emptyset\rhd_{PRF}\langle 1,t\rangle \quad \Theta\models\{\varphi_1\}u_\sharp\leftarrow\mathcal{O}_{challenge}(\langle 1,t\rangle)[\mathsf{sk};.]\{\varphi_2\}}{\mathcal{E},\Theta,C_1',(\varphi_1,\varphi_2)\vdash\emptyset\rhd_{PRF}u_\sharp}}{\mathcal{E},\Theta,C_1',(\varphi_0,\varphi_2)\vdash\emptyset\rhd_{PRF}(u_\sharp,\lambda_{\mathcal{G}})}$$

with:

$$\varphi_0=\{l_{hash}\mapsto[];l_{challenge}\mapsto[]\}$$
$$\varphi_1=\{l_{hash}\mapsto[x];l_{challenge}\mapsto[]\}$$
$$\varphi_2=\{l_{hash}\mapsto[x];l_{challenge}\mapsto[\langle 1,t\rangle]\}$$

$$C_0'=\{\}$$
$$C_1'=\{(\emptyset,\mathsf{sk},(),T^{glob}_{PRF,v},\top)\}$$
Note that $C_1'\cdot C_1'=C_1'$

Figure 7: Derivation of τ_2