Principles for pRHL Proofs of Cryptographic Protocols: How to Convince EasyCrypt that your Protocol is Secure

Manuel Barbosa*, François Dupressoir†, Cameron Low†, Charlotte Mylog‡, Sabine Oechsner§, Théo Vignon*¶
*Universidade do Porto (FCUP) and INESC TEC, Porto, PT
†University of Bristol, UK, {f.dupressoir,cameron.low.2018}@bristol.ac.uk
‡Norwegian University of Science and Technology (NTNU), NO, charlotte.mylog@ntnu.no
§Vrije Universiteit Amsterdam, NL, s.a.oechsner@vu.nl
¶Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, FR,
theo.vignon@lmf.cnrs.fr

***Abstract*—The EasyCrypt proof assistant has been used to successfully formalize security proofs for a wide variety of cryptographic primitives. However, attempts at formalizing objects with interactivity, such as protocols, have fared much worse. In this paper, we investigate (some of) the reasons for this difficulty. To identify which proof features contribute most to the complexity of formalization in a tool based on probabilistic relational Hoare logic (pRHL) such as EasyCrypt, we formalize a simple interactive key agreement protocol in three different ways. From these proof efforts, we then extract our insights. In particular, we argue that difficulties in formalizing the security of interactive protocols in the computational model arise from the fact that such proofs rely on both state and temporal invariants—the former to support cryptographic reasoning, and the latter to support reasoning about the protocol's structure. At the same time, EasyCrypt and similar pRHL-based approaches offer no native notion of events, traces or timestamps, forcing a user to find their own encoding of temporal properties. We discuss two possible encodings, state-first and trace-first, and their impact on the proof effort. More generally, we believe that this observation can help build new reasoning tools that can bridge the gap that currently exists between primitive-focused tools and protocol-focused tools.**

I. Introduction

Cryptography and cryptographic protocols are a key enabling technology for digital systems that are safe and respect their users' privacy. Giving independently-verifiable assurance of the quality of the cryptography that is deployed is also an important part of managing this technology responsibly—especially as the community prepares for significant changes.

Subtle differences in the security of underlying primitives (and in the definitions of those primitives) are known to make or break protocol security [CDM24], [CDF+21], yet no existing tool reliably formalizes security proofs for protocols while taking into account all the intricacies of the primitives they are instantiated with. So far, the only way to scale formal verification technology to machine-checked results [BBD+24], [BFK+13] has been to combine specialized tools, using trusted translations. This reliance on translations is precisely what we would like to move away from since it is unclear whether the composition of these results guarantees the required level of cryptographic security, and in particular whether the established security is at least as strong as that claimed and proved by hand in cryptography research papers.

EasyCrypt [BCLS15] appears to be a natural candidate for verifying protocols and their underlying primitives within a single framework. However, even though EasyCrypt supports combined proofs of protocols and primitives in principle, such proofs appear to be impractical in practice. This observation motivates the central question we address in this work: why does EasyCrypt struggle with protocol verification? That is, where exactly does the complexity lie in formally verifying protocol security proofs in the computational model? Although we focus on EasyCrypt, we believe our conclusions apply to any formal verification tool that relies on variants of Hoare logic, and particularly on probabilistic relational Hoare logics (pRHL) [BGZ09], to reason about game-based security proofs in the computational model. We perform our investigation as a case study based on a simple authenticated key exchange (AKE) protocol that allows to demonstrate typical protocol security arguments such as reductions from cryptographic assumptions, lazy/eager sampling arguments, and temporal reasoning.

A. Our contributions

Our main contributions are as follows:

- We identify a key source of difficulty in EasyCrypt formalization of proofs of cryptographic protocols: the encoding of temporal properties into pRHL in a way that is compatible with reasoning about random samplings;
- We propose an alternative to the typical state-oriented reasoning, to avoid the blow-up in invariant size inherent to interactivity: we encode time as ordering of events in a logical trace, rather than as protocol state machine changes.

While we believe these two contribute to the understanding of cryptographic proofs as complex proof objects, we also hope they will inform the development of existing and future reasoning support for protocol security in the computational model. Moreover, it seems that a systematic study of this type

of argument beyond pRHL (e.g. in Squirrel [BDJ+21]) would also yield valuable insights for proofs in those tools as well as future tool development.

B. Our Approach

The basis of our investigation is a simple 3-round AKE protocol called PSK-NSL [Lip22]. PSK-NSL is a pre-shared key (PSK) variant of the public key Needham-Schroeder-Lowe (NSL) protocol. Starting from a PSK psk_{AB}, the protocol allows two parties A and B to derive a fresh shared secret key, using an Authenticated Encryption with Associated Data (AEAD) scheme and a pseudo-random function (PRF). The AEAD scheme is used to exchange two nonces n_a and n_b, which are then used as PRF key to derive the shared key from other session parameters.

To pinpoint the security proof challenges for this particular protocol, we developed three different proofs in EasyCrypt[1] [BGHZ11], [BDG+13], a pRHL-based proof assistant for cryptographic security proofs in the code-based game-playing style. The full EasyCrypt code for our final proof[2] attempt is provided along with instructions on how to verify it. Once we had these proofs, we could compare and evaluate them to extract insights that generalize beyond PSK-NSL, which we summarize below. Sections IV–VII present our findings in more detail.

C. Our findings

The first attempt, a classic (ad-hoc) game hopping proof, and the second attempt, a proof in the style of State-Separating Proofs (SSP) [BDF+18], turned out to suffer from similar issues: The invariant style that we used seemed unsuitable for proving the game hops.

To understand what is going on, we have to take a step back. In the process of developing any game-hopping security proof in EasyCrypt, one needs to come up with suitable invariants that justify each game hop. A major challenge is always to identify these invariants. In our protocol security proof, the crucial arguments revolve around delaying random samplings to a later point in the protocol. Working with this type of temporal properties has not received much attention on the EasyCrypt side. To make matters worse, there is no native notion of traces and events, and hence the only way of working with temporal properties is to work in a suitable encoding. For the first two proofs, we went with a state-first encoding that tracks events as changes in the game state. The invariants contain essentially a full description of the protocol state machine, as well as the relationships between the internal states of the protocol state machine and the rest of the game states. A major downside of this encoding is the sheer size of invariants that quickly became challenging to handle.

In the subsequent third attempt, we went with a different encoding of temporal properties that turned out to lead to much more manageable invariants. The proof relies on the simple ordering properties of events in a trace to encode time, in contrast to using changing protocol states. In our case study, this trace-first encoding approach led to overall less complex invariants, both in terms of size and structure. It also reduced the effort involved in the iterative process of identifying and refining invariants while developing the proof.

While our case study is executed in EasyCrypt, we note that our finding are more general: The identified challenges are more broadly challenges of a pRHL-based approach, and so should apply to any tool in this family (e.g. EasyCrypt, SSProve [AHR+21], [LS25], FCF [PM15], Clutch [GAH+24] and CryptHOL [Loc16], [BLS20]). Tools such as VCV-io [TH24] or the recent plethora of vibed frameworks in Lean, not explicitly based on pRHL, may be worth investigating separately, although we note that most focus on direct semantic reasoning not typically amenable to manual formalization.

D. Outline

Section II introduces some background on authenticated key exchange as well as the primitives that our protocol PSK-NSL is built from. In Section III, we present the protocol PSK-NSL and outline how a pen-and-paper game hopping proof of AKE security would proceed. Section IV introduces our formalization approach and Section V discusses encoding temporal properties as key challenge for our proof effort. We evaluate two styles of encoding temporal invariants in pRHL (state first (Section VI) and trace first (Section VII)) and discuss their impact on our security proof. Finally, Section VIII discusses lessons learned.

II. Preliminaries

In this section, we introduce the cryptographic syntax and security models that we use throughout the rest of the paper.

A. Authenticated Key Exchange

Authenticated Key Exchange (AKE) protocols allow two parties to establish a shared session key that can then be used for secure communication, and is typically intended for short-term use and discarded after the session ends. In this work we consider the pre-shared secret setting that is common, for example, in mobile communication networks. Here, security is bootstrapped by providing each pair of parties with a pre-shared long-term symmetric key. The parties then use this pre-shared secret to securely establish new discardable session keys via an interactive protocol.

Our model considers a multi-user, multi-session setting where each instance is uniquely identifiable by a party's id a and instance number i. Two instances that fulfill the partnering notion described below are referred to as a session. The instance that sends the first message is given the role of *Initiator*, and the other instance becomes the *Responder*. We restrict our attention to three-round protocols and consider a standard (albeit modest) security goal: indistinguishability of session keys in the presence of a session key reveal oracle, but without long-term key corruption (and hence without any form of forward secrecy). Indeed, the protocol studied in this work is not secure under long-term secret key corruption: it was stripped down to the bare minimum to expose the challenges of protocol formal verification in EasyCrypt.

[1] https://easycrypt.info

[2] https://github.com/Cameron-Low/EC-KAP1

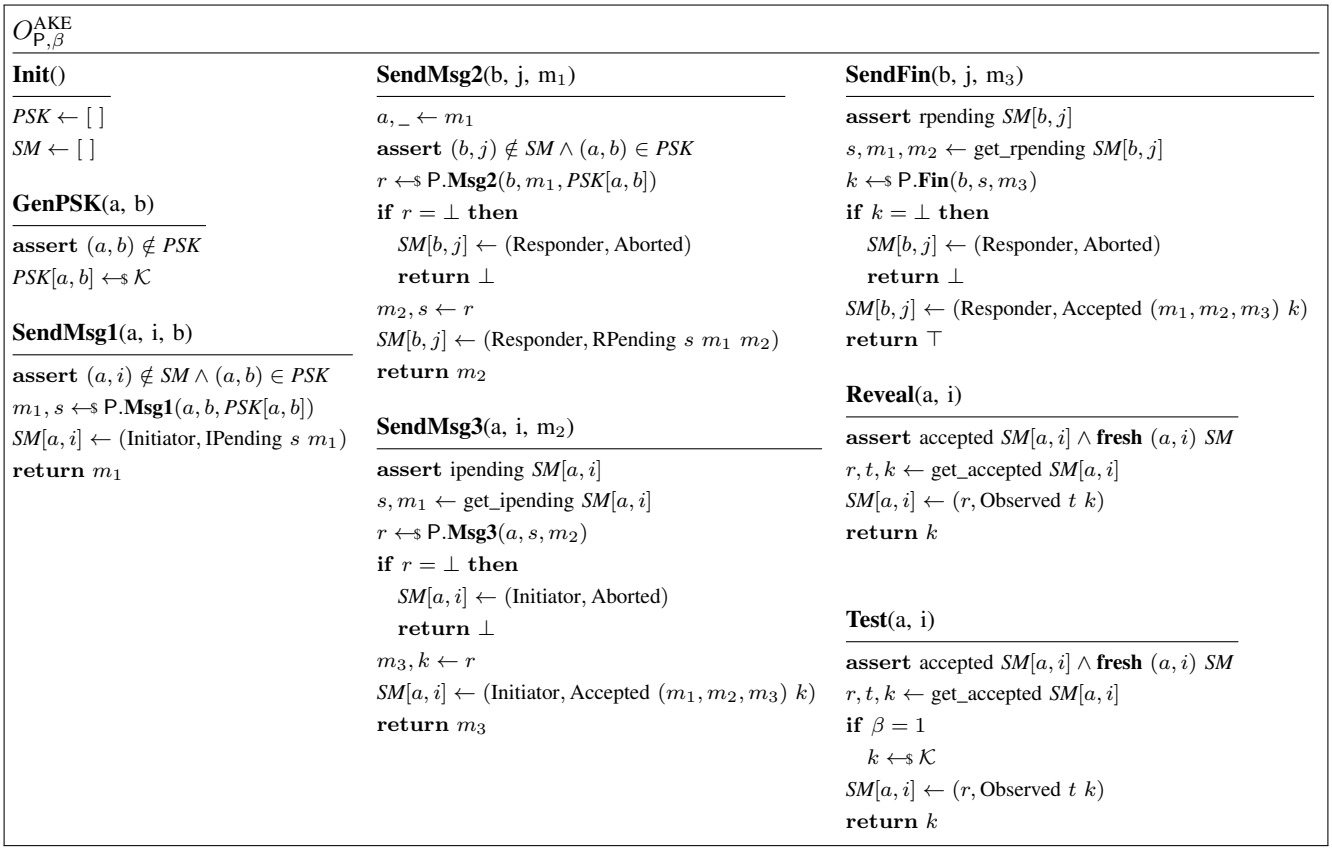


Fig. 1. The AKE oracles parametrised by an AKE protocol P and a real/ideal bit b

AKE Protocol: We consider three round AKE protocols specified by the following algorithms: $(\mathit{Msg1}, \mathit{Msg2}, \mathit{Msg3}, \mathit{Fin})$. The algorithms *Msg1* and *Msg3* are executed by the initiator, while *Msg2* and *Fin* are executed by the responder. The pre-shared long-term key sampled from the key space $\mathcal{K}$ is provided as input to the first two algorithms. All algorithms additionally take as input the relevant public information for the session, including the identities of the participating parties, incoming messages, and any required party state. Each algorithm may output a protocol message, an updated state, and a shared key, as applicable. The failure of any algorithm is signaled by an empty output.

Adversary Interactions: The interface of the protocol is exposed through the Send_* oracles, with explicit generation of the pre-shared long-term keys required through the GenPSK oracle. Moreover, the adversary has access to the Reveal and Test oracles; the former returns the shared key of a particular party instance, and the latter either reveals a shared key or simply returns a random key depending on the challenge bit. We place no restriction on the number of queries to these oracles. This shared key reveal is the only form of leakage permitted in our model. It exposes the primary challenges of the analysis of protocol security in the formalization setting. The full security model we employ is depicted in Fig. 1.

Protocol State: We keep track of the state of an instance via the following states:

- IPending s m_1: A pending initiator instance with state s that has sent m_1.
- RPending s m_1 m_2: A pending responder instance with state s that has received m_1 and sent m_2.
- Accepted t k: An accepted instance with protocol message transcript t and shared key k.
- Observed t k: An accepted instance that has been tested or revealed with transcript t and shared key k.
- Aborted: An aborted instance.

Partnering: Our partnering notion is defined in terms of the message transcript t that is publicly available. Two instances are considered partners if they have opposite roles and agree on the full transcript t of exchanged messages. Concretely, the initiator must have sent messages m_1 and m_3 and received m_2, while the responder must have received m_1 and m_3 and sent back m_2.

Freshness: To prevent trivial distinguishing attacks such as revealing and testing the same session, the Reveal and Test oracles need to have restrictions ensuring that only *fresh* sessions can be successfully queried. For our model, we consider an accepted instance (a, i) to be **fresh** with respect to a state *SM* if it has at most one other partner, and if that partner exists, its shared key has not been tested or revealed. This follows the filtering approach of [BCJ+25], in which responses to misbehaving adversarial queries are silenced in order to prevent trivial attacks.[3]

Security: The security of a protocol can be defined as:

[3]The case of an instance that has more than one partner is a breach of the mutual authentication property which must be analyzed separately. In our model we consider only confidentiality.

Definition 1 (AKE Security)**.** *A protocol* Π *is said to be AKE secure if the advantage of any ppt adversary* $\mathcal{A}$ *to distinguish a shared key belonging to a fresh session from a randomly sampled key is small.*

B. AEAD

Authenticated Encryption with Associated Data consists of a triple of algorithms (Gen, Enc, Dec). The Gen algorithm samples random keys from a key space $\mathcal{K}$. For encryption and decryption the Enc and Dec algorithms take a key, some additional authenticated data and the plaintext/ciphertext, respectively, as input and returns a ciphertext/plaintext.

We consider the standard indistinguishability security notion for AEAD [Rog02] extended to the multi-user setting. Both the real and ideal games provide three oracles

1) Gen, generate a key for the given handle if it has not already been generated.
2) Enc, encrypt a plaintext under the key corresponding to the handle, and the provided additional data.
3) Dec, decrypt a ciphertext under the key corresponding to the handle, and the provided additional data.

The real game performs encryption/decryption using the AEAD's encryption/decryption algorithm whereas the ideal game uses a decryption map with randomly generated ciphertexts. Decryption fails on all inputs that do not have an entry in the decryption map. An AEAD is secure if, for all ppt adversaries, the distinguishing advantage between these two games is small.

C. PRF

A pseudo-random function f maps a key from a key space $\mathcal{K}$ and input in $\mathcal{X}$ to an output space $\mathcal{Y}$. We consider the standard indistinguishability security notion for pseudo-random functions, again extended to the multi-user setting. Both the real and ideal games provide two oracles

1) Gen, generate a key for the given handle if it has not already been generated.
2) F, compute the value of the function f on the given input, under the key corresponding to the handle.

The real game computes the output of F using $\mathcal{F}_i$ whereas the ideal game builds a random function for each key generated/queried. A PRF is secure if, for all ppt adversaries, the distinguishing advantage between these two games is small.

III. Case Study: The PSK-NSL Protocol

As a vehicle for our investigation, we introduce a simple AKE protocol PSK-NSL. The protocol is designed to be conceptually minimal while still requiring a representative range of provable security arguments for its analysis. At the high-level, the protocol can be described as follows: Each party generates a nonce, and transmits it to the other party in encrypted form. The session key is derived using the PRF, keyed by the concatenation of the two nonces, and evaluated on (a part of) the transcript. To avoid replay attacks, the protocol messages are bound together via the associated data of the AEAD: each message can therefore be seen as a confirmation of the delivery of the previous one. As we will see, formally verifying the security of this protocol, is much more sophisticated than might be expected at first glance.

Why this Protocol?: While keeping the protocol reasonably simple, we still include complexity like interaction between instances and state that an instance has to keep track of. It also combines two cryptographic primitives, each requiring a proof step to apply their security property. Moreover, those two cryptographic primitives are intertwined, since the PRF is keyed with the nonces that are encrypted. Also note that the PRF assumption requires that the key is sampled *in one go* and remains secret to the adversary. This implies potential complexity since we use the concatenation of two nonces sampled by two different instances (and indeed two different parties) as key.

We first introduce the protocol itself, then provide a high-level outline of the pen-and-paper version of its security analysis. Looking ahead, this outline will form the basis for the mechanized proof attempts in EasyCrypt and our findings that will be discussed in the second half of the paper, starting in Section IV.

A. Protocol Overview

The PSK-NSL protocol is a simple 3-round key agreement protocol that can be seen as a pre-shared key variant of the public key Needham-Schroeder-Lowe protocol. The protocol assumes a pre-shared key psk_{AB} between the parties A and B, and establishes a new shared key k, using an AEAD scheme A and a PRF F. Figure 2 shows the protocol algorithms **PSK-NSL.Msg1**, **PSK-NSL.Msg2**, **PSK-NSL.Msg3**, **PSK-NSL.Fin** and how they are executed within one session between two parties A and B. Note that our AKE security notion will later consider this protocol for an arbitrary number of parties running an arbitrary number of sessions in parallel.

Assume that A acts as initiator and B as responder. Party A starts by sampling a fresh nonce n_A, encrypting it under the pre-shared key psk_{AB} with an AEAD scheme, and sending the resulting ciphertext to B. Party B can then decrypt, sample a nonce n_B, encrypt n_B, and send it to A. Upon decryption of n_B, A will compute the session key k by evaluating the PRF on a fixed input, using the concatenation $n_A||n_B$ as PRF key. As last step, A computes a key confirmation message by sampling and encrypting another nonce n_{ok}, and sending it to B. Finally, B also computes k by evaluating the PRF. Note that each ciphertext also includes, as associated data, the current session transcript to bind the message to the session.

B. Security

PSK-NSL achieves AKE security through an indistinguishability argument. Intuitively, the protocol is secure if no adversary can distinguish the real shared keys from uniformly sampled keys when making test queries. By the security of the PRF, we will know that if the PRF keys are random then the output will be too. Thus, we can use the security of AEAD scheme to ensure that the nonces are generated randomly just

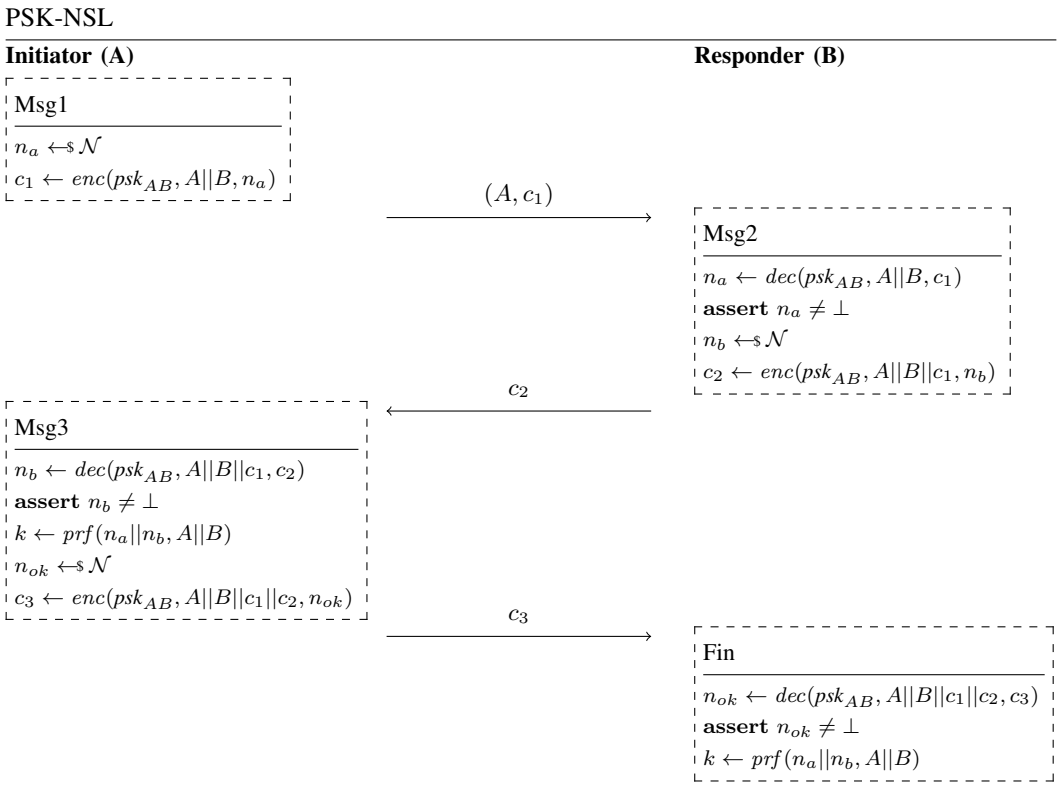


Fig. 2. Overview of key agreement protocol PSK-NSL.

before they are used as part of the PRF. Overall, our goal will be to establish the following theorem:

Theorem 1 (Security of PSK-NSL)**.** *Given an AEAD scheme* A *with ciphertext space* $\mathcal{C}$*, and a PRF* F*, the key agreement protocol PSK-NSL in Figure 2 is secure. That is, for any AKE adversary* $\mathcal{A}$ *that makes at most* q *queries to* $SendMsg1$*,* $SendMsg2$*, and* $SendMsg3$*, there exists reductions* $\mathcal{R}_0^{AEAD}, \mathcal{R}_1^{AEAD}, \mathcal{R}_0^{PRF}, \mathcal{R}_1^{PRF}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{AKE}_{PSK\text{-}NSL}(\mathcal{A}) \leq\ & \mathsf{Adv}^{AEAD}_{\mathsf{A}}(\mathcal{R}_0^{AEAD}) + \mathsf{Adv}^{AEAD}_{\mathsf{A}}(\mathcal{R}_1^{AEAD}) \\ & + \mathsf{Adv}^{PRF}_{\mathsf{F}}(\mathcal{R}_0^{PRF}) + \mathsf{Adv}^{PRF}_{\mathsf{F}}(\mathcal{R}_1^{PRF}) + \frac{2q^2}{|\mathcal{C}|}.\end{aligned}$$

The advantage of $\mathcal{A}$ is bounded by the sum of the advantages of distinguishing the AEAD and PRF games, and the probability of ciphertext collisions (each applied twice).

C. Proof Outline

The proof proceeds as a sequence of game transitions and consists of two parts. The first part starts from the real game and gradually idealizes the cryptography until reaching a game where the whole protocol is idealized, including the shared key. The second part then undoes this idealization (except for the shared key) to finally reach the ideal game, where the adversary again interacts with the real AKE oracles, except for the test queries. Throughout this paper, we focus only on describing the first half of the proof; the second part requires identical arguments to the first but applies them in reverse. Figure 3 gives an overview of the changes in each game hop.

$\underline{\mathsf{Game}_1 \to \mathsf{Game}_2}$*:* We replace the real encryption and decryption algorithms by idealized ones. Ideal encryption samples random ciphertexts and maintains a log of plaintext/ciphertext pairs which is used for ideal decryption. Indistinguishability is shown via a reduction $\mathcal{R}_0^{\text{AEAD}}$ to AEAD security. $\mathcal{R}_0^{\text{AEAD}}$ simulates the required AKE oracles but offloads to the AEAD oracles for pre-shared key generation, and encryption/decryption.

$\underline{\mathsf{Game}_2 \to \mathsf{Game}_3}$*:* We eliminate the possibility of ciphertext collisions. Using the failure event lemma, the distinguishing advantage can be bounded by the collision probability, which is a birthday bound on the total number of queries to $SendMsg1$, $SendMsg2$, and $SendMsg3$.

$\underline{\mathsf{Game}_3 \to \mathsf{Game}_4}$*:* Instead of each participant sampling their respective nonces individually in $SendMsg1$ and $SendMsg2$, we delay the sampling to $SendMsg3$ and sample both nonces at once. The nonces are stored in a map indexed by the transcript of the current session, and $SendFin$ can perform a lookup in the map. To show that the two games are identical, we need to argue that neither nonce's value is ever used until $SendMsg3$. In particular, this means showing that an initiator which successfully decrypts a second message will be the only possible partner for the associated responder: Since ciphertexts are unique, each protocol message sent is also unique. Each initiating/responding session can therefore be uniquely identified by the messages it sent, and nonce pairs can also jointly be indexed uniquely by the transcript of the session in which they are used.

$\underline{\mathsf{Game}_4 \to \mathsf{Game}_5}$*:* We replace real PRF queries with simulated ones, using a map and freshly sampled values in

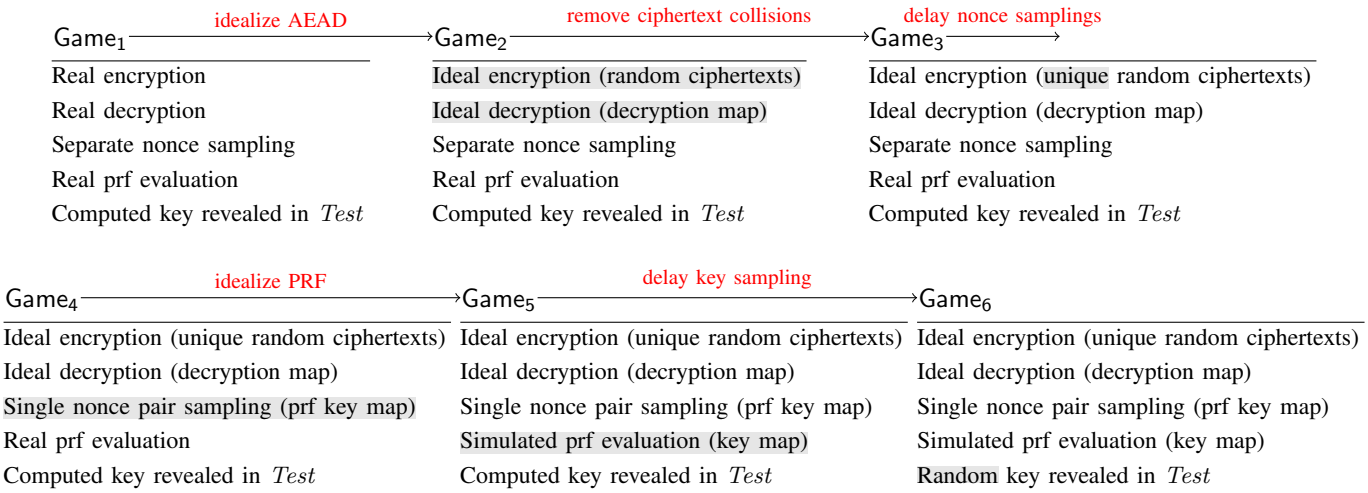


Fig. 3. Abstract view of the game structure

the range to store queries. Indistinguishability is shown via a reduction $\mathcal{R}_0^{\text{PRF}}$ to PRF security. $\mathcal{R}_0^{\text{PRF}}$ simulates the required AKE oracles with calls to the PRF oracles for generating nonce pairs, and the key.

$\mathsf{Game}_5 \rightarrow \mathsf{Game}_6$*:* We move the sampling of the session key to the test oracle. To show that the two games are identical, we need to show that a successful test query is guaranteed to be the first time that the shared key for that particular (possible) partnering will have been disclosed to the adversary. Since, as established previously, each initiator/responder can only have at most one partner, the adversary cannot have seen the session key before.

IV. Formalizing PSK-NSL Security

For the remainder of this paper, we will devote our time to exploring the main source of difficulty we encountered whilst formalizing the security proof of PSK-NSL— encoding temporal properties as state invariants. This section will outline the formalization process of Theorem 1 in EasyCrypt, as well as how our various proof attempts lead to the discovery of said source of difficulty.

A. Choice of Tool

Given our focus on security proofs using pRHL as the underlying foundation, EasyCrypt [BGHZ11] was a natural choice of tool. It is the most mature tool for reasoning about security in the computational model. Moreover, it has been used not only to prove security of cryptographic primitives (ML-KEM [AOB+24], SPHINCS+ [BDH+24]) but also for protocols (NAXOS [BCLS15], NTOR [DGLM26]).

B. Methodology

Our primary objective for this experiment was to identify the issues of formalizing protocols in pRHL-based tools. To achieve this, we decided to attempt multiple formalizations of the same security proof and then compare their processes and resulting proofs. Initially, we decided upon two distinct proof strategies: 1) a game hopping proof in the style of Bellare and Rogaway [BR06]; and 2) a proof in the style of State-Separating Proofs (SSP) [BDF+18]. The first approach follows the same path of historical EasyCrypt proofs, and provides an ideal baseline for our experiment. The SSP methodology of the second approach places an emphasis on highly modular security proofs, supporting the use of local reasoning. Since, SSP was originally developed in the context of miTLS, the F*-verified implementation of TLS 1.3 [FKS11], [BFK+13], [BFK+14], [DFK+17], we were optimistic about its benefit for other key exchange security proofs.

Whilst undertaking the two formalizations, we realized that, despite their structural differences, not only did both approach the proofs of key properties similarly, but they were also hindered by the same issues. This insight pushed us to attempt another strategy, proof 3), that aimed to differentiate itself not by overall proof structure — indeed this attempt also follows the style of Bellare and Rogaway — but by the approach taken in proving those properties. We will later see that the challenges faced in the formalizations are not determined by the actual proof approach.

In light of this fact, we will use proof 3) as a representative for all the approaches allowing us to describe more concretely these formalization challenges whilst maintaining brevity. A full description of the games that used in proof 3) can be found in Figure 8.

C. Formalizing Proofs in EasyCrypt

To appreciate where the difficulties arise, one needs to be familiar with the general process of formalizing security proofs in EasyCrypt.

EasyCrypt implements program logics (including probabilistic relational Hoare logic (pRHL) [BEG+15]) for reasoning about probabilistic, imperative programs such as cryptographic security games. It was designed specifically with the code-based game playing techniques of Bellare and Rogaway, as well as Shoup [Sho04] in mind. Thus, for proofs written in these styles the game and security definitions can be directly transcribed into EasyCrypt. In addition, EasyCrypt has also been used for security proofs in the SSP style [DKO22],

[DHL+25], wherein one can follow the straightforward embedding to transcribe definitions. This leaves the problem of formalizing the game hops.

Each game hop of an indistinguishability proof falls into one of three types of arguments:

- *Computational* - equivalence of games through reduction to a security assumption.
- *Statistical* - equivalence of games up to the occurrence of an unlikely event.
- *Perfect* - information theoretic equivalence of games.

Other than the soundness of reductions (an argument that also requires perfect indistinguishability), computational steps are simply applications of assumptions. However, to make statistical and perfect arguments in EasyCrypt, one relies on observational equivalence. For pRHL-based tools, one must show that the same adversary executed within two disjoint memories (one for each game in the hop) cannot distinguish the oracles of one game from the other. Concretely, one must prove that for each pair of oracles produces the same outputs when called with the same inputs. The proof often requires showing that the memories of the respective games maintain certain properties. In pRHL these properties are captured using state invariants.

State invariants constrain the set of reachable states the system can be in. In cryptographic proofs, we end up considering *'one-sided' state invariants* and *'two-sided' or relational state invariants* which relate the (distribution of) a pair of systems' states. (The latter can also be seen, in pRHL, as restricting the set of memories reachable by a product system.)

One-sided state invariants: These invariants are useful in expressing simple and somewhat general constraints on the state of one program. For example in Figure 3, when reasoning about the equivalence of Game_1 and Game_2, it is useful to establish—as a property of Game_1 alone—that if a responder instance is RPending with state s, then there must exist a key in the pre-shared key map.

Relational state invariants: These invariants relate the states of two programs (or constrain the state of a product program). Most commonly, they express equality (or equidistribution) of variables. As an example from our proofs: we show as part of the equivalence proofs between Game_4 and Game_5, and between Game_5 and Game_6, that the decryption maps are equal in both programs (unless the adversary can distinguish between the real and ideal PRF). Such invariants are also used widely in reasoning about reductions, where some state may move from experiment to oracles, while retaining its value.

A key part of any formalization is to derive a sound and sufficient set of invariants. In some cases, the set is trivial to state, perhaps only requiring equality of states, but the majority of the time, the full set of invariants is only discovered through trial and error. The common strategy follows a refinement process: an initial set of invariants is selected, and then iterated over when a failure in the proof is encountered. Generally speaking, the time/effort required to find a sufficient invariant set strongly correlates with the complexity of a given proof step. However, we shall see later, in Section VI and Section VII, that the strategy for selecting the initial set can cause a large variation in the required effort.

D. Challenges Formalizing PSK-NSL

Although the formalizations of each of the three proof approaches are different in structure, they follow the same trends of complexity—where a game hop in one proof approach is difficult to formalize, it is difficult in all. The same holds for those game hops that suffer no such difficulties. In the case of PSK-NSL, the computational and statistical indistinguishability arguments for the first, second, and fourth hops all fall into the latter category, with the necessary invariants essentially just capturing the equality of states. The remaining two hops, however, deserve more attention.

As outlined above, across all proof approaches the only two game hops that are non-trivial to formalize are $\mathsf{Game}_3 \rightarrow \mathsf{Game}_4$, and $\mathsf{Game}_5 \rightarrow \mathsf{Game}_6$. Both of these hops perform a similar operation: delaying the sampling of a value from one protocol state to another. The first hop requires that the session nonces are sampled as a pair when the initiating instance moves into an accepted state in $SendMsg3$ instead of individually in the initiating/responding instance's pending state in $SendMsg1$/$SendMsg2$. The second hop samples the session key when one of the instances of the session moves to the observed state in $Test$/$Reveal$ rather than just when the initiating instance moves to an accepted state in $SendMsg3$.

Recall from the proof sketch in Section III-C that these transformations are valid due to the fact that each sampling can be uniquely bound to an instance of the protocol, thus justifying the delay-until-use in each hop. However, since pRHL is inherently designed to handle synchronous samplings, it is not immediately clear how to formalize these arguments. Fortunately, there is a partial solution that alleviates some of the technical challenges posed by synchronous samplings and provides a mechanism for expressing some of these arguments in pRHL.

V. Delaying Sampling in pRHL

In this section, we explain the steps involved in the delaying of samplings in pRHL-based formalizations, and delve into the temporal properties that are fundamental to their soundness.

A. Randomness in pRHL

Before we look at the solution in detail, we will first attempt to provide some intuition for how it works. The first step is to re-evaluate how one thinks of randomness in pRHL. Whenever we encounter a sampling instruction in a pRHL proof, the value of the assigned variable is immediately 'fixed' either through coupling or simple execution. In particular, this means that no distinction is made between when the variable's value is determined and when it is used.[4] However, it should be clear that as long as we fix the variable's value before it is first observed, we could theoretically make this determination

[4]This is in contrast to other non-pRHL settings, such as Squirrel [BDJ+21] or CryptoVerif [Bla06], where fresh names and pre-sampled random tapes are involved.

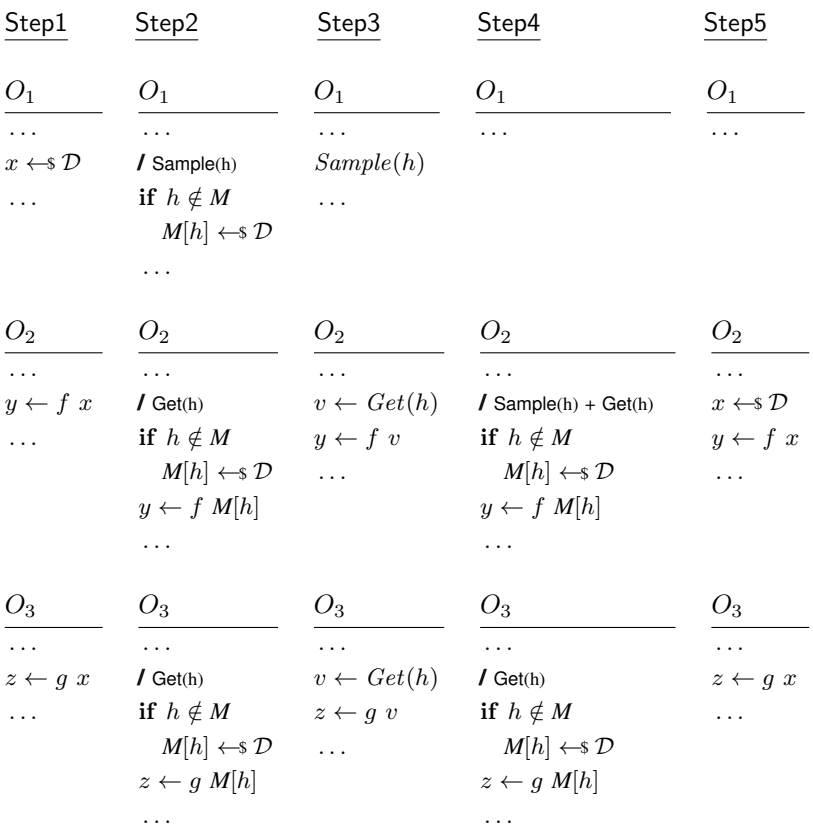


Fig. 4. Oversimplified proof sketch for moving samplings using handles.

at any point after the sampling instruction is encountered. In order to reason about this fact in pRHL, we must be able to reference the variable without needing to determine its value, for example by introducing handles. One such method introduces an abstraction that provides some indirection for samplings.

B. Abstracting Samplings

A central part of the strategy for delaying samplings in pRHL is the recovery of alignment between sampling instructions. This problem has already been addressed in prior work [ABB+19], and we omit the technical details here. The key result is that one can freely switch between the two implementations of sampling abstraction shown in Figure 5.

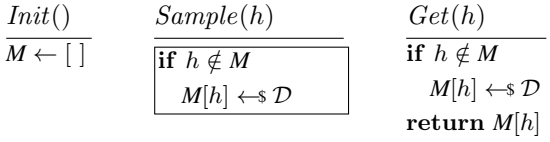


Fig. 5. The implementation of the lazy and eager sampling oracles. The boxed code is only present in the eager oracle.

The interface provides two oracles $Sample$ and Get that simulate the separation between when a sampling is performed and when its value requires determination. Note the use of handles h. For the eager oracle the determination happens upon first sampling, and for the lazy oracle determination happens upon first use.

C. Applying the Abstraction

In order to make use of the sampling abstraction, one must first choose an appropriate handle. Figure 4 illustrates a generic, albeit somewhat simplified, sequence of games for moving a sampling from one oracle to another. The second and third steps rely on free switching between the eager and lazy implementations of the abstraction. The first and last hops, however, require showing that the choice of handle is sound in the sense that each handle uniquely used in a call to $Sample$, and that said call must happen before any call to Get.

Formalizing these steps requires full understanding of the properties we need to encode as invariants.

D. Temporal Properties

The proof for soundness of handle choices requires proving *temporal* properties. These types of properties require a notion of time to relate the occurrence or non-occurrence of events in a given execution trace. For these game hops, we can more formally state the properties using the Sample and Get events annotated in Figure 4. Assuming we have access to an execution trace and timestamps (denoted by $\#i$), such properties are trivial to state.

- Uniquely Sampled Values.

$$\forall i\ h, \mathsf{Sample}(h)@\#i \implies \neg\exists j, \mathsf{Sample}(h)@\#j \wedge (\#j < \#i)$$

- Sound Value Retrieval.

$$\forall i\ h, \mathsf{Get}(h)@\#i \implies \exists j, \mathsf{Sample}(h)@\#j \wedge (\#j < \#i)$$

Game_3
Nonces sampled in $SendMsg1$ and $SendMsg2$
Nonces stored in State Map *SM* and Decryption Map *DM*
Nonces stored when sampled

Game_{3a}
Nonces sampled in $SendMsg1$ and $SendMsg2$
Nonces stored in Decryption Map *DM*
Nonces stored when sampled

Game_{3b}
Nonces sampled in $SendMsg1$ and $SendMsg2$
Nonces stored in Nonce Map *NM*
Nonces stored when sampled

Game_{3c}
Nonces sampled in $SendMsg3$ and $SendMsgF$
Nonces stored in Nonce Map *NM*
Nonces stored when sampled and not already present

Game_{3d}
Nonces sampled in $SendMsg3$
Nonces stored in Nonce Map *NM*
Nonces stored when sampled

Game_4
Nonces sampled in $SendMsg3$
Nonces stored in PRF Key Map *PRFKM*
Nonces stored when sampled

Fig. 6. Abstract overview of the changes between intermediate games of Game_3 and Game_4

In order to formalize these properties in EasyCrypt, we must find an alternate, stateful form for them, since there is no native way to encode them using pRHL. Of course, we could directly embed a notion of time and trace into each of the games using a new global state, but this only solves the issue of being able to state temporal formulae, not giving them stateful semantics. The core decision rests on how we encode timestamped events $\mathsf{Event}@\#i$ into purely stateful properties.

In particular, we will need to be able to distinguish two instances of the same event happening at different times, which requires our encoding to uniquely categorize the state changes corresponding to an event's occurrence. In the following sections, we will see the various methods for deciding how to bind these state changes to events.

VI. State-first Encodings

A critical aspect of encoding temporal events into stateful invariants lies in identifying changes in game state—the changes of values in the memory of an experiment—that can distinguish *when* each event is triggered. This leaves open the question of not only how many state changes we should keep track of but also which state changes we should track. Our first strategy, one followed by proof approaches 1) and 2), is to take cues from the object we are proving the security of. In this section, we will explain this strategy, and the effects it has on formalization using the $\underline{\mathsf{Game}_3 \rightarrow \mathsf{Game}_4}$ game hop as a running example.

A. The Protocol State Machine

Recall from Section IV-C that we use a refinement process for deriving a valid set of invariants that encode the desired properties needed in a given proof step. The first part of this process relies on choosing an intitial set of invariants that we can refine. As alluded to above, we will use the protocol as a starting point for encoding the required temporal properties.

Every protocol implicitly defines a state machine that is followed by all participating parties. For PSK-NSL, there are two possible initial pending states depending on whether you are an initiator or responder that can each transition to either an aborted state or an accepted state after executing the protocol. In the context of our security model (Figure 1), we have a further 'observed' state that may be reached upon a successful test or reveal query. Since we keep track of each instance's protocol state using the state map *SM*, we are able to directly encode this machine into an invariant.

$$
\begin{aligned}
&\forall h,\ \mathit{SM}[h] = \bot \\
&\quad \vee \exists s\ m_1, \mathit{SM}[h] = (\text{Initiator}, \text{IPending}\ s\ m_1) \\
&\quad \vee \exists s\ m_1\ m_2, \mathit{SM}[h] = (\text{Responder}, \text{RPending}\ s\ m_1 m_2) \\
&\quad \vee \exists r\ m_1\ m_2\ m_3\ k, \mathit{SM}[h] = (r, \text{Accepted}\ (m_1, m_2, m_3)\ k) \\
&\quad \vee \exists r\ m_1\ m_2\ m_3\ k, \mathit{SM}[h] = (r, \text{Observed}\ (m_1, m_2, m_3)\ k) \\
&\quad \vee \exists r, \mathit{SM}[h] = (r, \text{Aborted})
\end{aligned}
$$

Using this as a foundation for capturing a notion of time, we can further specify each state, and the relationships between their internal states and the rest of the game state. Exactly which relationships one needs to specify depends entirely on the proof step, and the implicit events involved, we are looking at so, for the sake of concreteness, we will look at how to approach the proof for the nonce delaying game hop.

B. A Stateful Approach to $\underline{\mathsf{Game}_3 \rightarrow \mathsf{Game}_4}$

Following the method outlined in Section V, we can split the game hop into a sequence of smaller game hops to isolate the real challenge of this step shown in Fig. 6. For brevity, we will keep the details required in formalizing the other steps to a minimum.

The game hops from Game_3 to Game_{3a} and Game_{3b} rely on very simple invariants - just equality between the states in each game. The hop from Game_{3b} to Game_{3c} corresponds precisely to the switching from eager to lazy sampling in the abstraction. On the other side, the hop from Game_{3d} to Game_4 only requires an equivalence between storing nonces individually (in a pair of maps) and storing them jointly (in a map of pairs).

Finally, we are left with the hop from Game_{3c} to Game_{3d}. Before we look at encoding the necessary temporal properties,

we need to consider the code of the games. The changes between the two games are relatively minor and isolated to just the $SendMsg3$ and $SendFin$ oracles. Figure 7 shows the exact differences that arise. In Game_{3d}, we eliminate the guards that prohibit overwriting an existing value in the nonce map in $SendMsg3$, and conversely we only retrieve the values from the nonce map in $SendFin$.

C. Embellishing the Machine

To begin to reason about this game hop, we need to keep track of the relationships between the internal states of the protocol state machine and the rest of the game states. These are fairly natural to express and are built on top of the existing invariant.

$$
\begin{aligned}
&\forall h,\ \mathit{SM}[h] = \bot \\
&\quad \vee \exists b\ a\ c_a, \\
&\qquad \mathit{SM}[h] = (\text{Initiator}, \text{IPending}\ (b, \bot, \dots)\ (a, c_a)) \\
&\qquad \wedge (a||b, c_a) \in \mathit{DM} \\
&\quad \vee \exists a\ c_a\ m_2, \\
&\qquad \mathit{SM}[h] = (\text{Responder}, \text{RPending}\ (a, \bot, \dots)\ (a, c_a)\ m_2) \\
&\qquad \wedge (a||h.1, c_a) \in \mathit{DM} \\
&\qquad \wedge (a||h.1||c_a, m_2) \in \mathit{DM} \\
&\quad \vee \exists r\ a\ c_a\ m_2\ m_3\ n_a\ n_b\ k, \\
&\qquad \mathit{SM}[h] = (r, \text{Accepted}\ ((a, c_a), m_2, m_3)\ k) \\
&\qquad \wedge \exists b, (a||b, c_a) \in \mathit{DM} \\
&\qquad\quad \wedge (a||b||c_a, m_2) \in \mathit{DM} \\
&\qquad\quad \wedge (a||b||c_a||m_2, m_3) \in \mathit{DM} \\
&\qquad\quad \wedge \mathit{NM}[(a||b, c_a)] = n_a \\
&\qquad\quad \wedge \mathit{NM}[(a||b||c_a, m_2)] = n_b \\
&\quad \vee \exists r\ a\ c_a\ m_2\ m_3\ n_a\ n_b\ k, \\
&\qquad \mathit{SM}[h] = (r, \text{Observed}\ ((a, c_a), m_2, m_3)\ k) \\
&\qquad \wedge \exists b, (a||b, c_a) \in \mathit{DM} \\
&\qquad\quad \wedge (a||b||c_a, m_2) \in \mathit{DM} \\
&\qquad\quad \wedge (a||b||c_a||m_2, m_3) \in \mathit{DM} \\
&\qquad\quad \wedge \mathit{NM}[(a||b, c_a)] = n_a \\
&\qquad\quad \wedge \mathit{NM}[(a||b||c_a, m_2)] = n_b \\
&\quad \vee \exists r, \mathit{SM}[h] = (r, \text{Aborted})
\end{aligned}
$$

The soundness proof for this invariant is rather simple given that we are merely describing the possible states after execution of any particular oracle. However, since we need to consider state changes we must also track the lack of a value's presence in the state. The only states where this is relevant is the initiator's pending state where we add two further relations.

$$(a||b, c_a) \notin \mathit{DM} \wedge \forall m_2, (a||b||c_a, m_2) \notin \mathit{DM}$$

The knock-on effect for adding these is that in order to maintain the soundness of the overall invariant, we must also add unicity property for intitiators.

$$\exists \hat{h}\ s\ a\ c_a, \mathit{SM}[\hat{h}] = \text{Initiator}, \text{IPending}\ s\ (a, c_a) \implies h = \hat{h}$$

Deriving this sufficient invariant set starts out being straightforward, since the execution of the protocol dictates much of the initial set. However, the sheer quantity of relationships that are described leaves room for simple error and mis-statements causing costly fixes to the proof. Moreover, maintaining the soundness proofs requires a substantial effort since they must be upheld any time there is a shift in protocol state.

As one can imagine, the more states a protocol has the larger the invariant we will need in order to describe it. This linear growth is deeply undesirable due to the large impact on the formalization. Thus, we need an alternative approach that might be less susceptible to this scaling.

VII. Trace-first Encodings

As we saw in the previous section, the state-first encoding strategy for temporal properties produces overly strong invariants. In order to reduce the burden in formalizing them, we explore a different strategy that relies on the simple ordering properties of events in a trace to encode time in contrast to using changing protocol states. In this section, we will describe this shift in focus and revisit our running example.

A. Implicit Traces

Although we have no native notion of traces and events in EasyCrypt, we can derive a trace from the state changes that occur alongside events. In particular, we can use monotonic, boolean state changes to keep track of whether or not a particular event would have occurred in the trace at a particular point in time. A simple example would be the existence of messages in the decryption map: the event that the first message has been sent can be directly encoded as $m_1 \in \mathit{DM}$.

Once a few of these trivial events have been identified, we can recover a notion of time by keeping track of an ordering between them. Continuing our example, we can derive the following simple orderings on messages.

1) If a second message has been sent, then so has a first.

$$\forall a\ b\ c_a\ m_2, (a||b||c_a, m_2) \in \mathit{DM} \implies (a||b, c_a) \in \mathit{DM}$$

2) If a third message has been sent, then so has a second.

$$\forall a\ b\ c_a\ m_2\ m_3, (a||b||c_a||m_2, m_3) \in \mathit{DM} \implies (a||b||c_a, m_2) \in \mathit{DM}$$

As we saw with the protocol state machine, these simple ordering properties provide a foundation upon which we can build. Next, we return to our ongoing example of formalizing the crux of the third game hop.

B. Exploiting the Nonce Maps

Recall from Figure 7 that we must show that both nonces are uniquely sampled in $SendMsg3$. Upon execution of the oracle in Game_{3c}, if the handle for either nonce was already in the nonce map, we can be certain that the underlying Sample event corresponding to that nonce's sampling, must have appeared in the trace prior to this point in time. Thus, we can use the presence or lack of presence of a handle as a

SendMsg3(a, i, m_2)

assert is_ipending $SM[a, i]$
$s, m_1 \leftarrow$ get_ipending $SM[a, i]$
$_, c_a \leftarrow m_1$
if $(a||b||c_a, m_2) \notin DM$ **then**
  $SM[a, i] \leftarrow$ (Initiator, Aborted)
  return $\bot$
$m_3 \leftarrow\!\$\ \mathcal{C}$
$DM \leftarrow DM \cup (a||b||c_a||m_2, m_3)$
$n_a \leftarrow\!\$\ \mathcal{N}$
$n_b \leftarrow\!\$\ \mathcal{N}$
if $(a||b, c_a) \notin NM$
  $NM[a||b, c_a] \leftarrow n_a$
if $(a||b||c_a, m_2) \notin NM$
  $NM[a||b||c_a, m_2] \leftarrow n_b$
$n_a, n_b \leftarrow NM[a||b, c_a], NM[a||b||c_a, m_2]$
$NM[a||b, c_a] \leftarrow n_a$
$NM[a||b||c_a, m_2] \leftarrow n_b$
$k \leftarrow$ **prf**$(n_a, n_b)(a, b)$
$SM[a, i] \leftarrow$ (Initiator, Accepted (m_1, m_2, m_3) k)
return msg_3

SendFin(b, j, m_3)

assert is_rpending $SM[b, j]$
$s, m_1, m_2 \leftarrow$ get_rpending $SM[b, j]$
$a, c_a \leftarrow m_1$
if $(a||b||c_a||m_2, m_3) \notin DM$ **then**
  $SM[b, j] \leftarrow$ (Responder, Aborted)
  return $\bot$
$n_a \leftarrow\!\$\ \mathcal{N}$
$n_b \leftarrow\!\$\ \mathcal{N}$
if $(a||b, c_a) \notin NM$
  $NM[a||b, c_a] \leftarrow n_a$
if $(a||b||c_a, m_2) \notin NM$
  $NM[a||b||c_1, m_2] \leftarrow n_b$
$n_a, n_b \leftarrow NM[a||b, c_a], NM[a||b||c_a, m_2]$
$n_a \leftarrow NM[a||b, c_a]$
$n_b \leftarrow NM[a||b||c_a, m_2]$
$k \leftarrow$ **prf**$(n_a, n_b)(a, b)$
$SM[b, j] \leftarrow$ (Responder, Accepted (m_1, m_2, m_3) k)
return $\top$

Fig. 7. Changes from Game_{3c} (shown solid) to Game_{3d} (shown dashed). The *SendMsg3* and *SendFin* oracles are identical in both games, except for the code in the solid boxes (Game_{3c} only) and dashed boxes (Game_{3d} only).

proxy for the sample event taking place previously. For similar reasons, we can also use the existence of a third message in the decryption map to derive this fact.

Pulling together a few more simple event orderings like 'if we have sampled the second nonce then we have sampled the first' we can construct a suitable initial invariant set.

- If we have sampled the second nonce, we have sampled the first.
$$\forall a\ b\ c_a\ m_2, (a||b||c_a, m_2) \in NM \implies (a||b, c_a) \in NM$$
- If we have sent a third message, we have sampled the second nonce.
$$\forall a\ b\ c_a\ m_2\ m_3, (a||b||c_a||m_2, m_3) \in DM \implies (a||b||c_a, m_2) \in NM$$
- If we have sampled the first nonce, we have sent the first message.
$$\forall a\ b\ c_a, (a||b, c_a) \notin DM \implies (a||b, c_a) \notin NM$$

Of course, this will not be a sufficient set, as we need an invariant that encodes the fact that a sampling of a nonce cannot have happened yet. It is at this stage that we turn back to using the protocol state as our guide. Indeed, for initiating instances that are in the pending state, we can be certain that the nonces have not yet been sampled. Hence, we can add a new invariant.

- Pending initiators have not sampled a nonce.
$$\forall a\ i\ r\ s\ \hat{a}\ c_a\ b, SM[a, i] = (r, \text{IPending}\ s\ (\hat{a}, c_a)) \implies (a||b, c_a) \notin NM$$

However, this invariant alone is not valid, and upon further refinement we discover the need for two more.

- Pending initiators have sent a first message.
$$\forall a\ i\ r\ s\ \hat{a}\ c_a\ b, SM[a, i] = (r, \text{IPending}\ s\ (\hat{a}, c_a)) \implies (a||b, c_a) \in DM$$
- Pending initiators are unique.
$$\forall h\ \hat{h}\ a\ s\ \hat{s}\ c_a, SM[h] = (\text{Initiator}, \text{IPending}\ s\ (a, c_a)) \wedge SM[\hat{h}] = (\text{Initiator}, \text{IPending}\ \hat{s}\ (a, c_a)) \implies h = \hat{h}$$

Comparing this final invariant set to that of the state-first strategy highlights just how important the encoding of time can be.

VIII. Discussion

We conclude the paper with a final discussion of the lessons learned from this case study, its impact beyond EasyCrypt, and some future work directions.

A. Protocol Security in EasyCrypt

The temporal properties that form the main challenge when formalizing the security of PSK-NSL are not unique to protocol proofs. As we have seen, simply delaying a sampling across oracles will give rise to them. However, what is unique to protocols is the large amount of interactivity that they bring. We cannot easily discern where in an execution trace we are from within an oracle call.

Moreover, EasyCrypt is not well suited to dealing with temporal properties in general. Although we have seen how to

embed a notion of traces into our proofs, the lack of any native trace reasoning within the program logics makes it challenging to encode these properties usefully. Our strategy in Section VII provides an intuition for how to approach this problem but is by no means a definitive and robust method. We encourage future research to explore this necessary interplay of reasoning styles at a more foundational level.

B. Generalizing to Other Tools

1) pRHL-Based Tools: Even though we executed this case study in a specific tool, EasyCrypt, we believe that the lessons learned are applicable to other tools in the same class of tools. As discussed throughout the paper, the challenges we faced seem inherent when attempting to encode temporal properties in a tool designed for reasoning about cryptographic experiments written as stateful, imperative probabilistic programs. If anything, EasyCrypt's existing automation and support for complex arguments, such as the method for delaying samplings described in Section V, has made our lives *easier* than what one may encounter in a tool without this kind of support.

2) Other Tools for the Computational Model: Given the inherent tension between reasoning about state properties and temporal properties present in protocol security proofs, it would be interesting to consider a similar exploration from the perspective of a tool where trace properties are native in the underlying logic such as Squirrel [BDJ+21] or CryptoVerif [Bla06].

Acknowledgments

We thank Benjamin Lipp for early discussions and for insights gained from an attempt at a CryptoVerif proof for NSL-PSK, as well as Andreas Hülsing for initially challenging our assumption that State-Separating Proofs were the right way to formalize protocol proofs. We also thank Adrien Koutsos, Justine Sauvage and Guillaume Scerri for the discussions around the Squirrel prover. We thank the Formosa Crypto project and the EasyCrypt team in particular for general discussions on machine-checked cryptography, and their continued support of the tools that make this research possible.

Cameron Low is fully funded by EPSRC Doctoral Training Partnership 2765784. Sabine Oechsner was supported by the Blockchain Technology Laboratory at the University of Edinburgh and funded by Input Output Global. This research was funded in part by Innovate UK SCHEME (Safety Critical Harsh Environment Micro-processing Evolution) project (10065634). This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

References

[ABB+19] José Bacelar Almeida, Cecile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Alley Stoughton, and Pierre-Yves Strub. Machine-checked proofs for cryptographic standards: Indifferentiability of Sponge and secure high-assurance implementations of SHA-3. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1607–1622, London, UK, November 11–15, 2019. ACM Press.

[AHR+21] Carmine Abate, Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Catalin Hritcu, Kenji Maillard, and Bas Spitters. SSProve: A foundational framework for modular cryptographic proofs in coq. In Ralf Küsters and Dave Naumann, editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–15, Virtual Conference, June 21–24, 2021. IEEE Computer Society Press.

[AOB+24] José Bacelar Almeida, Santiago Arranz Olmos, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Cameron Low, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Peter Schwabe, and Pierre-Yves Strub. Formally verifying Kyber - episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 384–421, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[BBD+24] Bruno Blanchet, Pierre Boutry, Christian Doczkal, Benjamin Grégoire, and Pierre-Yves Strub. CV2EC: Getting the best of both worlds. In *CSF 2024 Computer Security Foundations Symposium*, pages 279–294, Enschede, The Netherlands, July 8–12, 2024. IEEE Computer Society Press.

[BCJ+25] Chris Brzuska, Cas Cremers, Håkon Jacobsen, Douglas Stebila, and Bogdan Warinschi. *Falsifiability, Composability, and Comparability of Game-Based Security Models for Key Exchange Protocols*, pages 214–255. Springer Nature Switzerland, Cham, 2025.

[BCLS15] Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 689–718, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[BDF+18] Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. State separation for code-based game-playing proofs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 222–249, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.

[BDG+13] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier López, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

[BDH+24] Manuel Barbosa, François Dupressoir, Andreas Hülsing, Matthias Meijers, and Pierre-Yves Strub. A tight security proof for SPHINCS$^+$, formally verified. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 35–67, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

[BDJ+21] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. An interactive prover for protocol verification in the computational model. In *2021 IEEE Symposium on Security and Privacy*, pages 537–554, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[BEG+15] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanesco, and Pierre-Yves Strub. Relational reasoning via probabilistic coupling. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2015.

[BFK+13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[BFK+14] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer Berlin Heidelberg, Germany.

[BGHZ11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the work-

ing cryptographer. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany.

[BGZ09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 90–101. ACM, 2009.

[Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *2006 IEEE Symposium on Security and Privacy*, pages 140–154, Berkeley, CA, USA, May 21–24, 2006. IEEE Computer Society Press.

[BLS20] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *Journal of Cryptology*, 33(2):494–566, April 2020.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer Berlin Heidelberg, Germany.

[CDF+21] Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. In *2021 IEEE Symposium on Security and Privacy*, pages 1696–1714, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[CDM24] Cas Cremers, Alexander Dax, and Niklas Medinger. Keeping up with the KEMs: Stronger security notions for KEMs and automated analysis of KEM-based protocols. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1046–1060, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[DFK+17] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy*, pages 463–482, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[DGLM26] François Dupressoir, Kristian Gjøsteen, Cameron Low, and Charlotte Mylog. Formalizing and strengthening the security proof of ntor. *Cryptology ePrint Archive*, 2026.

[DHL+25] François Dupressoir, Andreas Hülsing, Cameron Low, Matthias Meijers, Charlotte Mylog, and Sabine Oechsner. How hard can it be to formalize a proof? - Lessons from formalizing CryptoBox three times in EasyCrypt. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part II*, volume 16246 of *LNCS*, pages 137–167, Melbourne, VIC, Australia, December 8–12, 2025. Springer, Singapore, Singapore.

[DKO22] François Dupressoir, Konrad Kohbrok, and Sabine Oechsner. Bringing state-separating proofs to EasyCrypt A security proof for cryptobox. In *CSF 2022 Computer Security Foundations Symposium*, pages 227–242, Haifa, Israel, August 7–10, 2022. IEEE Computer Society Press.

[FKS11] Cédric Fournet, Markulf Kohlweiss, and Pierre-Yves Strub. Modular code-based cryptographic verification. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011*, pages 341–350, Chicago, Illinois, USA, October 17–21, 2011. ACM Press.

[GAH+24] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G Haselwarter, Joseph Tassarotti, and Lars Birkedal. Asynchronous probabilistic couplings in higher-order separation logic. *Proceedings of the ACM on Programming Languages*, 8(POPL):753–784, 2024.

[Lip22] Benjamin Lipp. *Mechanized Cryptographic Proofs of Protocols and their Link with Verified Implementations. (Preuves mécanisées de protocoles cryptographiques et leur lien avec des implémentations vérifiées)*. PhD thesis, PSL University, Paris, France, 2022.

[Loc16] Andreas Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In *European Symposium on Programming*, pages 503–531. Springer, 2016.

[LS25] Markus Krabbe Larsen and Carsten Schürmann. Nominal state-separating proofs. In *CSF 2025 Computer Security Foundations Symposium*, pages 363–377, Santa Cruz, CA, USA, June 16–20, 2025. IEEE Computer Society Press.

[PM15] Adam Petcher and Greg Morrisett. The foundational cryptography framework. In *International Conference on Principles of Security and Trust*, pages 53–72. Springer, 2015.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107, Washington, DC, USA, November 18–22, 2002. ACM Press.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[TH24] Devon Tuma and Nicholas Hopper. VCVio: A formally verified forking lemma and fiat-shamir transform, via a flexible and expressive oracle representation. Cryptology ePrint Archive, Report 2024/1819, 2024.

APPENDIX

Game$_1$ - Game$_6$

Init()

$PSK \leftarrow [\,]$
$SM \leftarrow [\,]$
$bad \leftarrow [\,]$ / Game$_2$ - Game$_6$
$DM \leftarrow [\,]$ / Game$_2$ - Game$_6$
$NM \leftarrow [\,]$ / Game$_{3b}$ - Game$_{3d}$
$PRFKM \leftarrow [\,]$ / Game$_4$ - Game$_6$
$KM \leftarrow [\,]$ / Game$_5$ - Game$_6$

GenPSK(a, b)

assert $(a, b) \notin PSK$
$PSK[a, b] \leftarrow\!\!\$\ \mathcal{K}$

SendMsg1(a, i, b)

assert $(a, i) \notin SM \wedge (a, b) \in PSK$
$n_a \leftarrow\!\!\$\ \mathcal{N}$ / Game$_1$ - Game$_{3b}$
$c_a \leftarrow\!\!\$\ enc\ PSK[a, b]\ (a||b)\ n_a$ / Game$_1$ - Game$_{1a}$
$c_a \leftarrow\!\!\$\ \mathcal{C}$ / Game$_2$ - Game$_6$
$bad \leftarrow bad \vee \exists d, (d, c_a) \in DM$ / Game$_2$ - Game$_6$
if bad **then** / Game$_3$ - Game$_6$
  return $\perp$ / Game$_3$ - Game$_6$
$DM[a||b, c_a] \leftarrow n_a$ / Game$_2$ - Game$_{3a}$
$DM[a||b, c_a] \leftarrow \top$ / Game$_{3b}$ - Game$_6$
$NM[a||b, c_a] \leftarrow n_a$ / Game$_{3b}$
$s \leftarrow (b, PSK[a, b], n_a, c_a)$ / Game$_1$
$s \leftarrow (b, \perp, n_a, \perp)$ / Game$_{1a}$ - Game$_3$
$s \leftarrow (b, \perp, \perp, \perp)$ / Game$_{3a}$ - Game$_6$
$SM[a, i] \leftarrow$ (Initiator, IPending $s\ (a, c_a)$)
return c_a

SendMsg2(b, j, m_1)

$a, c_a \leftarrow m_1$
assert $(b, j) \notin SM \wedge (a, b) \in PSK$
$n_a \leftarrow dec\ PSK[a, b]\ (a||b)\ c_a$ / Game$_1$ - Game$_{1a}$
$n_a \leftarrow DM[a||b, c_a]$ / Game$_2$ - Game$_{3a}$
if $n_a = \perp$ **then** / Game$_1$ - Game$_{1a}$
if $(a||b, c_a) \notin DM$ **then** / Game$_2$ - Game$_6$
  $SM[b, j] \leftarrow$ (Responder, Aborted)
  return $\perp$
$n_b \leftarrow\!\!\$\ \mathcal{N}$ / Game$_1$ - Game$_{3b}$
$m_2 \leftarrow\!\!\$\ enc\ PSK[a, b]\ (a||b||c_a)\ n_b$ / Game$_1$ - Game$_{1a}$
$m_2 \leftarrow\!\!\$\ \mathcal{C}$ / Game$_2$ - Game$_6$
$bad \leftarrow bad \vee \exists d, (d, m_2) \in DM$ / Game$_2$ - Game$_6$
if bad **then** / Game$_3$ - Game$_6$
  return $\perp$ / Game$_3$ - Game$_6$
$DM[a||b||c_a, m_2] \leftarrow n_b$ / Game$_2$ - Game$_{3a}$
$DM[a||b||c_a, m_2] \leftarrow \top$ / Game$_{3b}$ - Game$_6$
$NM[a||b||c_a, m_2] \leftarrow n_b$ / Game$_{3b}$
$s \leftarrow (a, PSK[a, b], n_a, n_b, c_a, m_2)$ / Game$_1$
$s \leftarrow (\perp, \perp, n_a, n_b, \perp, \perp)$ / Game$_{1a}$ - Game$_3$
$s \leftarrow (\perp, \perp, \perp, \perp, \perp, \perp)$ / Game$_{3a}$ - Game$_6$
$SM[b, j] \leftarrow$ (Responder, RPending $s\ m_1\ m_2$)
return m_2

SendMsg3(a, i, m_2)

assert ipending $SM[a, i]$
$(b, psk, n_a, c_a), m_1 \leftarrow$ get_ipending $SM[a, i]$
$a, c_a \leftarrow m_1$ / Game$_{1a}$ - Game$_6$
$psk \leftarrow PSK[a, b]$ / Game$_{1a}$ - Game$_6$
$n_b \leftarrow dec\ psk\ (a||b||c_a)\ m_2$ / Game$_1$ - Game$_{1a}$
if $n_b = \perp$ **then** / Game$_1$ - Game$_{1a}$
if $(a||b||c_a, m_2) \notin DM$ **then** / Game$_2$ - Game$_6$
  $SM[a, i] \leftarrow$ (Initiator, Aborted)
  return $\perp$
$n_{ok} \leftarrow\!\!\$\ \mathcal{N}$
$m_3 \leftarrow\!\!\$\ enc\ psk\ (a||b||c_a||m_2)\ n_{ok}$ / Game$_1$ - Game$_{1a}$
$m_3 \leftarrow\!\!\$\ \mathcal{C}$ / Game$_2$ - Game$_6$
$DM[a||b||c_a||m_2, m_3] \leftarrow n_{ok}$ / Game$_2$ - Game$_{3a}$
$DM[a||b||c_a||m_2, m_3] \leftarrow \top$ / Game$_{3b}$ - Game$_6$
$n_a \leftarrow DM[a||b, c_a]$ / Game$_{3a}$
$n_b \leftarrow DM[a||b||c_a, m_2]$ / Game$_2$ - Game$_{3a}$
$n_a \leftarrow\!\!\$\ \mathcal{N}$ / Game$_{3c}$ - Game$_6$
$n_b \leftarrow\!\!\$\ \mathcal{N}$ / Game$_{3c}$ - Game$_6$
if $(a||b, c_a) \in NM$ **then** / Game$_{3c}$
  $NM[a||b, c_a] \leftarrow n_a$ / Game$_{3c}$ - Game$_{3d}$
if $(a||b||c_a, m_2) \in NM$ **then** / Game$_{3c}$
  $NM[a||b||c_a, m_2] \leftarrow n_b$ / Game$_{3c}$ - Game$_{3d}$
$n_a \leftarrow NM[a||b, c_a]$ / Game$_{3b}$ - Game$_{3c}$
$n_b \leftarrow NM[a||b||c_a, m_2]$ / Game$_{3b}$ - Game$_{3c}$
$PRFKM[a||b||c_a||m_2, m_3] \leftarrow (n_a, n_b)$ / Game$_4$ - Game$_{5a}$
$k \leftarrow prf\ (n_a||n_b)\ (a, b)$ / Game$_1$ - Game$_4$
$k \leftarrow\!\!\$\ \mathcal{K}$ / Game$_5$ - Game$_6$
$KM[m_1, m_2, m_3] \leftarrow k$ / Game$_5$ - Game$_{5a}$
$k \leftarrow \perp$ / Game$_{5a}$
$SM[a, i] \leftarrow$ (Initiator, Accepted $(m_1, m_2, m_3)\ k$)
return m_3

SendFin(b, j, m_3)

assert rpending $SM[b, j]$
$(a, psk, n_a, n_b, c_a, c_b), m_1, m_2 \leftarrow$ get_rpending $SM[b, j]$
$a, c_a \leftarrow m_1$ / Game$_{1a}$ - Game$_6$
$psk \leftarrow PSK[a, b]$ / Game$_{1a}$ - Game$_6$
$n_{ok} \leftarrow dec\ psk\ (a||b||c_a||c_b)\ m_3$ / Game$_1$ - Game$_{1a}$
if $n_{ok} = \perp$ **then** / Game$_1$ - Game$_{1a}$
if $(a||b||c_a||m_2, m_3) \notin DM$ **then** / Game$_2$ - Game$_6$
  $SM[b, j] \leftarrow$ (Responder, Aborted)
  return $\perp$
$n_a \leftarrow DM[a||b, c_a]$ / Game$_{3a}$
$n_b \leftarrow DM[a||b||c_a, m_2]$ / Game$_{3a}$
$n_a \leftarrow\!\!\$\ \mathcal{N}$ / Game$_{3c}$
$n_b \leftarrow\!\!\$\ \mathcal{N}$ / Game$_{3c}$
if $(a||b, c_a) \in NM$ **then** / Game$_{3c}$
  $NM[a||b, c_a] \leftarrow n_a$ / Game$_{3c}$
if $(a||b||c_a, m_2) \in NM$ **then** / Game$_{3c}$
  $NM[a||b||c_a, m_2] \leftarrow n_b$ / Game$_{3c}$
$n_a \leftarrow NM[a||b, c_a]$ / Game$_{3b}$ - Game$_{3c}$
$n_b \leftarrow NM[a||b||c_a, m_2]$ / Game$_{3b}$ - Game$_{3c}$
$PRFKM[a||b||c_a||m_2, m_3] \leftarrow (n_a, n_b)$ / Game$_4$ - Game$_{5a}$
$k \leftarrow prf\ (n_a||n_b)\ (a, b)$ / Game$_1$ - Game$_4$
$k \leftarrow KM[m_1, m_2, m_3]$ / Game$_5$
$k \leftarrow \perp$ / Game$_{5a}$ - Game$_6$
$SM[b, j] \leftarrow$ (Responder, Accepted $(m_1, m_2, m_3)\ k$)
return $\top$

Reveal(a, i)

assert accepted $SM[a, i] \wedge$ fresh (a, i) SM
$r, t, k \leftarrow$ get_accepted $SM[a, i]$
$k' \leftarrow\!\!\$\ \mathcal{K}$ / Game$_{5b}$ - Game$_6$
if $t \in KM$ **then** / Game$_{5b}$
  $KM[t] \leftarrow k'$ / Game$_{5b}$ - Game$_6$
$k \leftarrow KM[t]$ / Game$_{5a}$ - Game$_6$
$SM[a, i] \leftarrow (r,$ Observed $t\ s)$
return k

Test(a, i)

assert accepted $SM[a, i] \wedge$ fresh (a, i) SM
$r, t, k \leftarrow$ get_accepted $SM[a, i]$
$k' \leftarrow\!\!\$\ \mathcal{K}$ / Game$_{5b}$ - Game$_6$
if $t \in KM$ **then** / Game$_{5b}$
  $KM[t] \leftarrow k'$ / Game$_{5b}$ - Game$_6$
$k \leftarrow KM[t]$ / Game$_{5a}$ - Game$_{5c}$
$k \leftarrow\!\!\$\ \mathcal{K}$ / Game$_6$
$SM[a, i] \leftarrow (r,$ Observed $t\ k)$
return k

Fig. 8. Game definitions for proof 3). Each instruction is annotated with the game in which it appears. Instructions without annotation appear in all games.