Verification Methods for the Computationally Complete Symbolic Attacker Based on Indistinguishability

GERGEI BANA, University of Missouri, USA and University of Luxembourg, Luxembourg
ROHIT CHADHA and AJAY KUMAR EERALLA, University of Missouri, USA
MITSUHIRO OKADA, Keio University, Japan

In recent years, a new approach has been developed for verifying security protocols with the aim of combining the benefits of symbolic attackers and the benefits of unconditional soundness: the technique of the computationally complete symbolic attacker of Bana and Comon (BC) [8]. In this article, we argue that the real breakthrough of this technique is the recent introduction of its version for indistinguishability [9], because, with the extensions we introduce here, for the first time, there is a computationally sound symbolic technique that is syntactically strikingly simple, to which translating standard computational security notions is a straightforward matter, and that can be effectively used for verification of not only equivalence properties but trace properties of protocols as well. We first fully develop the core elements of this newer version by introducing several new axioms. We illustrate the power and the diverse use of the introduced axioms on simple examples first. We introduce an axiom expressing the Decisional Diffie-Hellman property. We analyze the Diffie-Hellman key exchange, both in its simplest form and an authenticated version as well. We provide computationally sound verification of real-or-random secrecy of the Diffie-Hellman key exchange protocol for multiple sessions, without any restrictions on the computational implementation other than the DDH assumption. We also show authentication for a simplified version of the station-to-station protocol using UF-CMA assumption for digital signatures. Finally, we axiomatize IND-CPA, IND-CCA1, and IND-CCA2 security properties and illustrate their usage. We have formalized the axiomatic system in an interactive theorem prover, Coq, and have machine-checked the proofs of various auxiliary theorems and security properties of Diffie-Hellman and station-to-station protocol.

CCS Concepts: • **Security and privacy → Logic and verification**; *Formal security models*;

Additional Key Words and Phrases: Authentication, secrecy, computational model, Dolev-Yao model, computational soundness, first-order logic

ACM Reference format:
Gergei Bana, Rohit Chadha, Ajay Kumar Eeralla, and Mitsuhiro Okada. 2019. Verification Methods for the Computationally Complete Symbolic Attacker Based on Indistinguishability. *ACM Trans. Comput. Logic* 21, 1, Article 2 (October 2019), 44 pages.
https://doi.org/10.1145/3343508

Gergei Bana was partially supported by the ERC Consolidator Grant CIRCUS (683032) and by the National Research Fund (FNR) of Luxembourg under the Pol-Lux project VoteVerif (POLLUX-IV/1/2016). Rohit Chadha and Ajay Kumar Eeralla were partially supported by NSF CNS 1314338 and NSF CNS 1553548. Mitsuhiro Okada was partially supported by JSPS KAKENHI Grants No. 17H02263, No. 17H02265, and No. JSPS-AYAME (2016–2018).
Authors' addresses: G. Bana, R. Chadha, and A. K. Eeralla, Department of Electrical Engineering and Computer Science, 201 Naka Hall, Columbia, MO 65211, USA; emails: bana@math.upenn.edu, chadhar@missouri.edu, ae266@mail.missouri.edu; M. Okada, Department of Philosophy, Keio University, 2-15-45 Mita, Minato-ku, Tokyo, 108-8345, Japan; email: mitsu@abelard.flet.keio.ac.jp.

1529-3785/2019/10-ART2 $15.00
https://doi.org/10.1145/3343508

1 INTRODUCTION

Security protocols are analyzed with respect to "attacker models," which formalize the capabilities of the attacker. There are primarily two approaches to rigorously model the capabilities of the attacker. The first approach, inspired by the theory of *computational complexity*, essentially says that a protocol is secure if an *attacker*, modeled as a polynomially bounded probabilistic Turing machine, can break the security property only with *negligible probability*. The second approach, inspired by the theory of *logic and programming languages*, assumes perfect black-box cryptography and nondeterministic *symbolic* computation by the attacker. It is common to call the former model the *computational model* while the latter the *Dolev-Yao model*.

The computational model generally provides far stronger security guarantees than the Dolev-Yao model. However, proofs of security in the computational model tend to be complex and error-prone. The Dolev-Yao model on the other is simpler and intuitive, and several tools are available for automatically proving security in the Dolev-Yao model, such as ProVerif [14], Scyther [21], and Tamarin [28].

Given that proofs in the computational model tend to be long and error-prone, it is desirable to have machine-assisted proofs. Two main research directions have been considered in the literature to achieve this goal. The first one is to establish *computational soundness results* (see References [2, 4], for example), which show that under certain conditions, the Dolev-Yao model is fully abstract with respect to the computational one and thus it is sufficient to analyze protocols in the Dolev-Yao model. Another approach is to carry out symbolic proofs of correctness directly in the computational model with the help of formal provers as is the case with CryptoVerif [15] and EasyCrypt [12]. Both of these approaches have limitations. Computational soundness results require strong assumptions on the computational implementation, calling into question their utility. Furthermore, when considering additional primitives, one has to establish the soundness results for the whole system again. As the proofs of *computational soundness results* are rather complex, this imposes a significant burden. Because of these issues, although once a research direction receiving much attention, it has largely been abandoned by now. The efforts of most researchers currently go into developing tools that work directly in the computational model. However, the current state-of-the-art formal provers are often not able to complete the proofs in computational model, even for secure protocols. When the provers fail to complete the proof, it is not clear if the failure is due to a protocol flaw or due to the limitations of the prover.

A third approach advocated by Bana and Comon (BC) in References [8, 9] (and developed in References [5, 9, 10, 26]), uses first-order logic for the verification of complexity-theoretic properties of protocols and for discovering attacks if verification fails. The notion of *symbolic attacker* is kept, but instead of specifying a restricted list of function symbols (with specific interpretation such as pairing, encryption) the attacker can use as is the case in the Dolev-Yao model, the BC technique only specifies the facts that the attacker cannot violate. More precisely, attacker computation in the BC technique for indistinguishability [9] is represented by function symbols f_1, f_2, which can satisfy anything that does not violate the aforementioned facts. These facts come from the nature of probabilistic polynomial time computation and the underlying cryptographic assumptions. The facts are called axioms and form a recursive set of first-order formulas. Without any axiom, the symbolic attacker is allowed to do anything (i.e., attacker messages can satisfy any property), and all protocols are insecure. Adding axioms limits the attacker and makes verification of protocols possible. Once verification is done with a set of axioms, the protocol is secure with respect to any implementation that satisfies the axioms. For the rest of the article, we shall call this approach the BC technique.

When verifying security of protocols with the BC technique, one tries to prove the security goal expressed as a first-order formula from axioms using first-order inference rules. If one manages to

build a complete proof tree, then a proof of security in the computational model follows, as long as the axioms are computationally valid. However, when a proof tree does not exist, a branch of the incomplete tree that does not reduce to an axiom yields an abstract first-order model—an Herbrand model—with an abstract domain together with an interpretation of function symbols on this domain, including the function symbols representing adversarial messages, which a computational attacker may perform to launch an attack (e.g., in Reference [5] a new attack on the NSL protocol found with this technique was presented). Furthermore, all computational attacks are symbolically accounted for as any computational attack yields a model in which both the axioms and negation of the security goal are true. It is for the latter reason that Reference [8] coined the term *computationally complete symbolic attacker* for the symbolic attacker in this approach. Whether the found attack corresponds to an actual PPT attack—that is, the symbolic interpretation of adversarial function symbols can be PPT executed—depends on how complete the axioms are.

Thus, the BC technique overcomes significant limitations of the Dolev-Yao technique when it comes to computational soundness, while maintaining its simplicity. As compared to the aforementioned tools working directly in the computational model, if a proof fails in the BC framework, then a possible attack is constructed. If the proof succeeds, then it provides a set of formulas, without any implicit assumptions, that, if satisfied by the implementation, result in a secure protocol. Furthermore, this approach is not stricken by the *commitment problem* (see, e.g., Reference [25]), an issue with all other symbolic verification techniques.

While the initial papers on the computationally complete symbolic attacker focussed on deducibility properties, Reference [9] extended the approach to *indistinguishability* properties: two protocols Π_1, Π_2 are said to be *computationally indistinguishable* if for each probabilistic polynomial-time attacker, the difference in the probability that it outputs 1 when interacting with Π_1 and the probability that it outputs 1 when interacting with Π_2 is negligible. Several standard security properties are modeled as indistinguishability properties. These include strong flavors of confidentiality, privacy, anonymity, real-or-random secrecy.

Our contributions. While Reference [9] sets up the framework (first-order logic based on a predicate $\sim$ representing indistinguishability of terms) needed to model the computationally complete symbolic attacker for indistinguishability properties, the set of axioms introduced therein were only sufficient to prove one session of a simple protocol they considered. In this article, we introduce several further axioms, grouped in two: *core axioms* that are independent from the protocols and primitives and just formalize indistinguishability, random generation etc, and *cryptographic axioms* that formalize hardness assumptions (such as the decisional Diffie-Hellman assumption), and standard security assumptions about cryptographic primitives (such as CPA, CCA, etc.). We then use the axioms to verify fairly complex protocols by hand, and with Coq.

One of the important contributions of this work is to axiomatize the if _ then _ else _ constructor. We illustrate through a number of perhaps surprising examples in Section 7 the power of the axioms. They are basic, general axioms, *not designed* with any particular protocol on our minds. We present a completeness theorem for the axiomatization of if _ then _ else _. The full set of core axioms presented is likely not complete, but we do believe that they cover most situations relevant for protocol equivalence in general. The axioms are independent. They are also modular and addition of the axioms will not destroy their validity.

The next group of main contributions are the axiomatization of the Decisional Diffie-Hellman (DDH) assumption, the verification of secrecy of the Diffie-Hellman (DH) protocol for multiple sessions, the axiomatization of security of digital signatures, and the verification of authentication of an authenticated DH protocol. The formalization of real-or-random secrecy and authentication in the BC framework for equivalence properties is also our novel contribution.

We then show how real-or-random secrecy [3] of the exchanged key can be formalized and verified in the BC framework. This is carried out for the case when each agent can participate in two sessions (both allowed to play the initiator as well as the responder role). Our proof can be easily generalized to any bounded number of sessions and more than two parties.

We axiomatize *existential unforgeability against chosen message attacks (UF-CMA)* [24] of digital signatures, and show that the technique and our axioms can also be used to verify a trace property: *authentication*. Toward this end, we present an authenticated version of the DH protocol, which is a simplified version of the station-to-station protocol (STS), formalize authentication in the current framework and verify authentication from the responder's view. Generalizations to arbitrary bounded number of sessions and agents are a straightforward matter in this case as well.

We have formalized the syntax and the axiomatic system in Coq, an interactive theorem-prover [29], and utilized the axiomatization to obtain machine-checked proofs of various auxiliary theorems, the derivation of the DDH assumption for three participants from the DDH assumption of two participants, real-or-random secrecy of the DH protocol, and authentication of the STS protocol in Coq. For the DH and STS protocol, we consider only one session each of responder and initiator to keep the formula small. All the machine-checked proofs are available at Reference [22]. While we were able to translate the proofs directly into Coq, a challenge was posed by the equality predicate = (actually an abbreviation constructed from ∼) defined between terms (see Section 2), which represents not identity but equality that can fail with negligible probability, and which serves as a congruence in the logic. The native equality relation of Coq forces two terms of our syntax to be equal (in the sense of native Coq equality relation) if and only if they are syntactically identical. This assumption would be unsound for our semantics, where syntactic difference does not necessarily entail inequality. Hence, we define our own equality relation and use morphisms [29] to model the defined relation as a congruence relation. This allowed us to reuse native Coq tactics such as *rewrite* and *replace* seamlessly.

Our final contribution is a common axiomatization of IND-CPA, IND-CCA1, and IND-CCA2 security properties of encryptions. An axiom for IND-CPA was also presented in Reference [9], the two are equivalent. The IND-CCA1 and IND-CCA2 axioms are new. We also illustrate with an example how to use the IND-CCA2 axiom.

We would like to highlight that in the BC framework based on indistinguishability, the standard cryptographic notions seem to translate very smoothly to axioms, such that the axiom is sound *if and only if* the computational security property holds. This is indicated by our DDH, UF-CMA, IND-CPA, IND-CCA1, and IND-CCA2 axioms. Note *further* that although the authors of [9] designed this technique for indistinguishability properties, it can also be conveniently used for trace properties such as authentication.

Related Work. There are other attempts in the literature for computationally sound analysis of Diffie-Hellman-based protocols. Most notably, in Reference [23], the authors explain how in computational PCL they can only verify Diffie-Hellman-based protocols as long as terms are non-malleable. For that reason, they need to sign their Diffie-Hellman terms for the verification of secrecy. We do not need any such assumption. CryptoVerif [15] has also been used to verify signed Diffie-Hellman key exchange protocols. AKE protocols have been verified using the EasyCrypt [12] proof assistant, with Computational DH assumption [11].

CryptoVerif [15] is a fully automated tool that specifies protocols in a process-based notation. It attempts to carry out proofs of protocols using in-built game transformations. It is, however, not guaranteed to terminate. Failure of termination does not mean that the protocol is insecure and in that case the tool does not provide an attack. Strengthening assumptions on cryptographic primitives and parsing of terms may help in termination.

EasyCrypt [12] is a semi-automated tool, and protocols are specified as programs in WHILE-style language with random assignments. The reasoning engine in uses EasyCrypt uses Probabilistic Relational Hoare logic as its reasoning engine to relate the distributions induced by two programs. The security proofs in EasyCrypt are carried out using game transformations. However, the needed game transformations have to be explicitly specified by the user. One advantage of EasyCrypt is that it can explicitly manipulate security parameter and probabilities. Thus, it can give estimates on concrete security. However, like CryptoVerif, it does not find attacks.

Both CryptoVerif and EasyCrypt verify protocols for a polynomial number of sessions (in the security parameter). BC technique for indistinguishability, however, reasons only for a fixed number of sessions. However, as opposed to game transformation techniques such as CryptoVerif and EasyCrypt, BC technique has explicit attacker representation in the form of function symbols f_1, f_2, and so on, which are interpreted in the symbolic attacker model when there is no proof of security, and provide an attack on the protocol. CryptoVerif and EasyCrypt do not have explicit attacker representation, and a failure of proof does not provide an attack. Furthermore, the reasoning can be carried out in first-order logic without resorting to probabilities, which often simplifies the proofs.

Most of the materials in this article (except Section 11 on formalization in Coq) first appeared in the e-print publication Reference [6], and the axioms motivated later works, such as References [7, 18, 20, 27].

2 SYNTAX

We shall follow closely the notation in Reference [9]. We summarize the salient features of the syntax and the semantics of the logic below, and the reader is referred to Reference [9] for details. We shall introduce the additional syntax needed for the Diffie-Hellman key exchange through running examples.

2.1 Terms

Let S be a finite set of *sorts* that includes at least the sorts bool and message. $\mathcal{X}$ is an infinite set of *variable symbols*, each coming with a sort $s \in S$.

The set $\mathcal{N}$ of *names* (for random seeds) is an infinite set of symbols that are treated as functions symbols of arity 0 and sort message. The set of elements of $\mathcal{N}$ shall be interpreted as random bit strings.

In addition, we assume a (fixed) set of function symbols, $\mathcal{F}$. Each element of $\mathcal{F}$ has a type, which is an element of the set $S^* \times S$. When $\text{type}(f) = (s_1, \ldots, s_n, s)$, we also write $f : s_1 \times \cdots \times s_n \to s$ and call n the *arity* of f. We assume that $\mathcal{F}$ includes at least the basic symbols such as **0**, ***true***, ***false***, EQ(_, _), and if _ then _ else _ with the typing rules as follows:

- **0** : message represents the empty message.
- Boleans
  true : bool ***false*** : bool.
- Polymorphic equality test
  $\text{EQ}(\_,\_) : \begin{array}{l} \text{message} \times \text{message} \to \text{bool} \\ \text{bool} \times \text{bool} \to \text{bool}. \end{array}$
- Polymorphic conditional branching
  $\text{if \_ then \_ else \_} : \begin{array}{l} \text{bool} \times \text{message} \times \text{message} \to \text{message} \\ \text{bool} \times \text{bool} \times \text{bool} \to \text{bool}. \end{array}$

We also use the following abbreviation:

(1) $\text{not}(b) \stackrel{\text{def}}{\equiv} \text{if } b \text{ then } \textbf{\textit{false}} \text{ else } \textbf{\textit{true}}.$

Example 2.1. Since in this work we consider the Diffie-Hellman key exchange, we shall need exponentiation. Although not necessary for the DH protocol, we also include pairing and projection functions as it shall be useful for combining messages. Accordingly, we shall also include in $\mathcal{F}$ the following function symbols:

$$\begin{array}{lrcl} \mathsf{exp}\_(\_,\_): & \mathsf{message} \times \mathsf{message} \times \mathsf{message} & \to & \mathsf{message} \\ \langle \_,\_ \rangle: & \mathsf{message} \times \mathsf{message} & \to & \mathsf{message} \\ \pi_1(\_), \pi_2(\_): & \mathsf{message} & \to & \mathsf{message}. \end{array}$$

The subscript of exp takes G that stands for a cyclic group, the first argument g is for an element of the group, and the second argument is the exponent. We shall use the abbreviations $g^a \stackrel{\text{def}}{\equiv} \mathsf{exp}_G(g,a)$ and $g^{ab} \stackrel{\text{def}}{\equiv} (g^a)^b$. Note that we do not write G explicitly in the abbreviation.

We also need function symbols for the algorithms that generate groups, their generators, and exponents so that their distributions satisfies the DDH assumption. We introduce

- generate group specification and generator
  $\mathsf{ggen}(\_) : \mathsf{message} \to \mathsf{message}$
- generate exponent (the "r" stands for ring)
  $\mathsf{r}(\_) : \mathsf{message} \to \mathsf{message}$.

The function symbol ggen is for the algorithm that generates a pair consisting of the description of a cyclic group G and a generator g of the group. We shall write $\mathsf{G}(\_)$ for $\pi_1(\mathsf{ggen}(\_))$, and $\mathsf{g}(\_)$ for $\pi_2(\mathsf{ggen}(\_))$. r is to denote the algorithm that generates an exponent randomly. We specified them as being given on message, but honest agents shall only apply them on names in $\mathcal{N}$.

We shall use the variables $g, g_1, g_2, \ldots$ to abbreviate a term of the form $\mathsf{g}(x)$. We shall also use the variables $a, b, c, d, \ldots$ to abbreviate terms of the form $\mathsf{r}(x)$ in the exponents of g's. When $\mathsf{ggen}(\_)$ and $\mathsf{r}(\_)$ are applied correctly on $\mathcal{N}$ then they will satisfy the DDH assumption.

Furthermore, as we want to consider multiple sessions as well, we need a way for the attacker to instruct an agent to start a new session. For this, we shall include

- start new session: $\mathsf{new} : \mathsf{message}$
- specify action: $\mathsf{act}(\_) : \mathsf{message} \to \mathsf{message}$
- message body: $\mathsf{m}(\_) : \mathsf{message} \to \mathsf{message}$.

A call by the attacker for starting a new session is then expressed by $\mathsf{EQ}(\mathsf{act}(x), \mathsf{new}) = 1$ for input variable x. The main message part, where g^a is supposed to come from the other agent is $\mathsf{m}(x)$.

Equational theory: We also postulate that the above functions satisfy the following equations:

$$\pi_k(\langle x_1, x_2 \rangle) = x_k \text{ for } k = 1, 2; \qquad g^{ab} = g^{ba}.$$

At this point, these are just strings of characters, but they will become axioms as EqTheo in Table 1.

While $\mathcal{F}$ contains function symbols necessary for the system and also those representing cryptographic primitives, an additional set of function symbols $\mathcal{G}$ represents adversarial computation. $\mathcal{G}$ contains countably many symbols: for every natural number n at least one whose type is $\mathsf{message}^n \to \mathsf{message}$. In the BC technique, a message from the adversary always has the form $f(t_1, \ldots, t_n)$, where $f \in \mathcal{G}$ and $t_1, \ldots, t_n$ are the messages from honest agents sent earlier. As in this technique, there is no Dolev-Yao-type pattern matching, the adversarial message is *not a term* created from function symbols in $\mathcal{F}$. As we shall see later, $f \in \mathcal{G}$ is allowed to satisfy any property that does not contradict the axioms.

We shall also use $\vec{f} : s_1, \ldots, s_n \to s'_1, \ldots, s'_m$ to denote a vector of functions $\{f_i : s_1, \ldots, s_n \to s'_i\}_{i=1}^m$.

We assume that $\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X}$ are disjoint. Terms are built using $\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X}$, following the sort discipline: for each $s \in S$, let $T_s(\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X})$ be the smallest set such that

- if $n \in \mathcal{N}$, then $n \in T_{\text{message}}(\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X})$, and if $x \in \mathcal{X}$ has sort s, then $x \in T_s(\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X})$ and
- if $f : s_1 \times \cdots \times s_n \to s$ is a symbol of $\mathcal{F} \cup \mathcal{G}$, and $t_1 \in T_{s_1}(\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X}), \ldots, t_n \in T_{s_n}(\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X})$, then $f(t_1, \ldots, t_n) \in T_s(\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X})$.

We do not have implicit coercion: a term of sort bool cannot be seen (also) as a term of sort message.

Example 2.2. Given $\mathcal{F}$ as defined in Example 2.1, variables g, a, and $f \in \mathcal{G}$, then

$$\mathsf{if}\ \mathsf{EQ}(\mathtt{act}(f(g)), \mathsf{new})\ \mathsf{then}\ g^a\ \mathsf{else}\ \mathbf{0}$$

is a term of sort message. This means that if the message $f(g)$ (computed from the public g) from the adversary indicates the start of a new session, then a new a is generated and g^a is sent.

Remark 1. To display the formulas more concisely, we use the abbreviations

$$\begin{aligned} &\mathsf{if}\ b\ \mathsf{then}\ t \overset{\text{def}}{\equiv} \mathsf{if}\ b\ \mathsf{then}\ t\ \mathsf{else}\ \mathbf{0}, \\ &b_1\ \&\ b_2 \overset{\text{def}}{\equiv} \mathsf{if}\ b_1\ \mathsf{then}\ b_2\ \mathsf{else}\ \boldsymbol{false} \\ &b_1\ \|\ b_2 \overset{\text{def}}{\equiv} \mathsf{if}\ b_1\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ b_2. \end{aligned}$$

2.2 Formulas

We have for every sequence of sorts $s_1, \ldots, s_n$ a *predicate symbol* that takes $2 \times n$ arguments of sort $(s_1 \times \cdots \times s_n)^2$, which we write as $t_1, \ldots, t_n \sim u_1, \ldots, u_n$ (overloading the notations for the predicate symbols with different types). $t_1, \ldots, t_n \sim u_1, \ldots, u_n$ represents *computational indistinguishability* of the two sequences of terms $t_1, \ldots, t_n$ and $u_1, \ldots, u_n$.

Our set of formulas, which will be used both for axioms and security properties are *first-order formulas built on the above atomic formulas.*

When we do not explicitly quantify variables, we shall mean universal quantification. Furthermore, any first order formula $\forall g.\theta[g]$ is an abbreviation for $\forall x.\theta[\mathsf{g}(x)]$, $\forall a.\theta[a]$ is an abbreviation for $\forall x.\theta[\mathsf{r}(x))]$ (similarly for b).

Example 2.3. The following is a formula: $g, g^a, g^b, g^{ab} \sim g, g^a, g^b, g^c$.

This is actually almost the form of the Decisional Diffie-Hellman assumption, except that we will need to make sure that g, a, b and c are independently, correctly generated. This shall be discussed when we state our DDH axiom.

We shall use the following abbreviation:

- $x = y \overset{\text{def}}{\equiv} \mathsf{EQ}(x, y) \sim \boldsymbol{true}$. This represents *computational equality* of terms. The choice of the equality symbol for this abbreviation is motivated by the fact that this functions as equality: it is a congruence relation with respect to our syntax (see Remark 3 and Section 5).

Remark 2. We extend the abbreviation = to sequences of terms as follows:

$$t_1, \ldots, t_n = u_1, \ldots, u_n \text{ if and only if } \forall i.\ t_i = u_i.$$

3 SEMANTICS

In the BC technique, two semantics are considered for the first-order formulas. The first is computational semantics: For the formulas to be interpreted computationally and to be able to consider their computational validity, computational semantics is needed. The other is abstract first-order semantics. In this technique, a symbolic attack means consistency of the axioms with the negation of the security property, which is equivalent to the existence of an abstract first-order model satisfying the axioms and the negation of the security property. We follow closely the definitions given in Reference [9].

3.1 Abstract First-order Interpretation

As usual in first-order logic: The *domain D* of the interpretation can be anything (and in our case it has subsets of bools and messages). Function symbols can be freely interpreted as some functions over this domain, predicates again freely interpreted as relations over this domain. Interpretation of logical constants, namely, negation, entailment, conjunction, disjunction, and quantification are fixed to be the usual Tarskian interpretation.

3.2 Computational Interpretation

A computational model $\mathcal{M}^c$ is a particular first-order model in which the domain consists of probabilistic polynomial-time algorithms. The interpretation of function symbols is limited to polynomial-time algorithms such that the outputs of the machines interpreting the domain elements are inputs to these algorithms. The interpretation of the predicate $\sim$ is fixed to be computational indistinguishability of probability distributions. More precisely, it is defined the following way:

1. The domain of sort message (denoted by D_{message} or D_{m} in short) is the set of deterministic Turing machines $\mathcal{A}$ equipped with an input (and working) tape and two extra tapes (that are used for the random inputs). All tapes carry bit strings only, the additional tapes contain infinitely long randomly generated bit strings. We require that the computation time of $\mathcal{A}$ is polynomial in the worst case w.r.t the input (not the content of the extra tapes). One of the extra tapes is shared by honest agents for drawing random values, while the other is used by the attacker when it draws random values. We write $\mathcal{A}(w; \rho_1; \rho_2)$ for the output of the machine $\mathcal{A}$ on input w with extra tape contents ρ_1, ρ_2.

The domain of sort bool is the set of such machines whose output is in $\{0, 1\}$. We denote this by D_{bool} (or D_{b} in short).

2. A function symbol $f \in \mathcal{F} \cup \mathcal{G}$, $f : s_1 \times \cdots \times s_n \to s$ is interpreted as a mapping $[\![f]\!] : D_{s_1} \times \cdots \times D_{s_n} \to D_s$ defined by some polynomial time (deterministic) Turing machine $\mathcal{A}_f$ such that for $(d_1, \ldots, d_n) \in D_{s_1} \times \cdots \times D_{s_n}$:

- If $f \in \mathcal{F}$, then $[\![f]\!](d_1, \ldots, d_n)$ is the machine that on input w and extra tapes ρ_1, ρ_2, outputs

$$[\![f]\!](d_1, \ldots, d_n)(w; \rho_1; \rho_2) := \mathcal{A}_f(d_1(w; \rho_1; \rho_2), \ldots, d_n(w; \rho_1; \rho_2))$$

  In other words, the way $[\![f]\!]$ acts on $(d_1, \ldots, d_n)$ is that we compose the machine $\mathcal{A}_f$ with the machines $d_1, \ldots, d_n$. Note that the machine $\mathcal{A}_f$ cannot use directly the tapes ρ_1, ρ_2.
- If $g \in \mathcal{G}$, then $[\![g]\!](d_1, \ldots, d_n)$ is the machine such that, on input w and extra tapes ρ_1, ρ_2, it outputs

$$[\![g]\!](d_1, \ldots, d_n)(w; \rho_1; \rho_2) := \mathcal{A}_g(d_1(w; \rho_1; \rho_2), \ldots, d_n(w; \rho_1; \rho_2); \rho_2).$$

Note that the machine $\mathcal{A}_g$ cannot use directly the tape ρ_1: the interpretations of function symbols in $\mathcal{G}$ are chosen by the attackers who cannot use directly the possibly secret values generated from ρ_1, but may use extra randomness from ρ_2.

- For all computational models, we require fixed interpretations of the following function symbols:
  - $[\![\textbf{\textit{true}}]\!]$ is the algorithm in D_b outputting 1 on all inputs.
  - $[\![\textbf{\textit{false}}]\!]$ is the algorithm in D_b outputing 0 on all inputs.
  - $[\![\mathbf{0}]\!]$ is the algorithm in D_m terminating with no output.
  - if _ then _ else _ is interpreted as a function $[\![\text{if \_ then \_ else \_ }]\!] : D_\mathsf{b} \times D_\mathsf{m} \times D_\mathsf{m} \to D_\mathsf{m}$ such that on the triple $(d, d_1, d_2) \in D_\mathsf{b} \times D_\mathsf{m} \times D_\mathsf{m}$, it gives the algorithm $[\![\text{if \_ then \_ else \_ }]\!](d, d_1, d_2)$ with
    $$[\![\text{if \_ then \_ else \_ }]\!](d, d_1, d_2)(w; \rho_1; \rho_2) := \begin{cases} d_1(w; \rho_1; \rho_2) & \text{if } \ d(w; \rho_1; \rho_2) = 1 \\ d_2(w; \rho_1; \rho_2) & \text{if } \ d(w; \rho_1; \rho_2) = 0 \end{cases}$$
    If $d_1, d_2 \in D_\mathsf{b}$, then $[\![\text{if \_ then \_ else \_ }]\!](d, d_1, d_2) \in D_\mathsf{b}$.
  - $\mathsf{EQ}(\_, \_)$ is interpreted as the function $[\![\mathsf{EQ}(\_, \_)]\!] : D_\mathsf{m} \times D_\mathsf{m} \to D_\mathsf{b}$ such that $[\![\mathsf{EQ}(\_, \_)]\!](d_1, d_2)$ is the algorithm with
    $$[\![\mathsf{EQ}(\_, \_)]\!](d_1, d_2)(w; \rho_1; \rho_2) := \begin{cases} 1 & \text{if } \ d_1(w; \rho_1; \rho_2) = d_2(w; \rho_1; \rho_2) \\ 0 & \text{if } \ d_1(w; \rho_1; \rho_2) \neq d_2(w; \rho_1; \rho_2) \end{cases}$$

3. A name $n \in \mathcal{N}$ is interpreted as the machine $[\![n]\!] = \mathcal{A}_n$ that, given a word of length η, extracts a word of length $p(\eta)$ from ρ_1 for some non-constant polynomial p. This machine does not use ρ_2. Different names extract disjoint parts of ρ_1, hence they are *independently generated*. We assume that p is the same for all names, that is the semantics is parametrized by this p. This way, all names are drawn independently, uniformly at random from $\{0,1\}^{p(\eta)}$.

4. Given a term t, an assignment σ of the free variables of t, taking values in the corresponding domains D_s, a security parameter η and a sample ρ (ρ is a pair $\rho_1; \rho_2$), $[\![t]\!]^\sigma_{\eta,\rho}$ is defined recursively as:

- for a variable x, $[\![x]\!]^\sigma_{\eta,\rho} := (x\sigma)(1^\eta; \rho)$ (the output of the algorithm $x\sigma$ on $1^\eta; \rho$, or, equivalently, the output of the machine interpreting x on the input 1^η, with random tapes ρ),
- for a name n, $[\![n]\!]^\sigma_{\eta,\rho}$ is the output of the machine $\mathcal{A}_n$ on 1^η and tape ρ,
- for a function symbol $f \in \mathcal{F}$, $[\![f(t_1, \ldots, t_n)]\!]^\sigma_{\eta,\rho} := [\![f]\!]([\![t_1]\!]^\sigma_{\eta,\rho}, \ldots, [\![t_n]\!]^\sigma_{\eta,\rho})$,
- for a function symbol $g \in \mathcal{G}$, $[\![g(t_1, \ldots, t_n)]\!]^\sigma_{\eta,\rho} := [\![g]\!]([\![t_1]\!]^\sigma_{\eta,\rho}, \ldots, [\![t_n]\!]^\sigma_{\eta,\rho}, \rho_2)$.

5. The indistinguishability predicate $\sim$ is interpreted as computational indistinguishability $\approx$ of sequences of elements in D of the same length. That is: $d_1, \ldots, d_n \approx d'_1, \ldots, d'_n$ iff for any polynomial time Turing machine $\mathcal{A}$,

$$|\mathbf{Prob}\{\rho : \ \mathcal{A}(d_1(1^\eta; \rho), \ldots, d_n(1^\eta; \rho); \rho_2) = 1\} - \mathbf{Prob}\{\rho : \ \mathcal{A}(d'_1(1^\eta; \rho), \ldots, d'_n(1^\eta; \rho); \rho_2) = 1\}|$$

is negligible in η. In particular, given an assignment σ of free variables in D_s, and an interpretation $[\![\cdot]\!]$ of the function symbols as above, $\sim$ is interpreted as the relation $\approx$ between sequences of the same length, which is defined as follows: $[\![t_1, \ldots, t_n]\!] \approx [\![u_1, \ldots, u_n]\!]$ iff for any polynomial time Turing machine $\mathcal{A}$,

$$|\mathbf{Prob}\{\rho : \ \mathcal{A}([\![t_1]\!]^\sigma_{\eta,\rho}, \ldots, [\![t_n]\!]^\sigma_{\eta,\rho}; \rho_2) = 1\} - \mathbf{Prob}\{\rho : \ \mathcal{A}([\![u_1]\!]^\sigma_{\eta,\rho}, \ldots, [\![u_n]\!]^\sigma_{\eta,\rho}; \rho_2) = 1\}|$$

is negligible in η. We write $\mathcal{M}^c, \sigma \models t_1 \ldots t_n \sim u_1 \ldots u_n$, and say that $\mathcal{M}^c, \sigma$ satisfies $t_1 \ldots t_n \sim u_1 \ldots u_n$. Satisfaction of compound formulas is defined from satisfaction of atomic formulas as usual in first-order logic. We write $\mathcal{M}^c, \sigma \models \theta$ if $\mathcal{M}^c, \sigma$ satisfies the first-order formula θ in the above sense. If $\vec{x}$ is the list of free variables in θ, then $\mathcal{M}^c \models \theta$ stands for $\mathcal{M}^c \models \forall \vec{x}.\theta$. A formula is *computationally valid* if it is satisfied in all computational models.

Remark 3. It is easy to see that with these definitions, satisfaction of the equality abbreviation $t_1, \ldots, t_n = u_1, \ldots, u_n$ by $\mathcal{M}^c, \sigma$ turns out to hold if and only if

$$\mathbf{Prob}\{\rho : [\![t_1]\!]^{\sigma}_{\eta,\rho} = [\![u_1]\!]^{\sigma}_{\eta,\rho}, \ldots, [\![t_n]\!]^{\sigma}_{\eta,\rho} = [\![u_n]\!]^{\sigma}_{\eta,\rho}\}$$

is overwhelming (that is, it is negligibly different from 1). In other words, the interpretations of $t_1, \ldots, t_n$ and of $u_1, \ldots, u_n$ may only differ by negligible probability. Note that this is much stronger than indistinguishability, which requires only the distributions of the interpretations of $t_1, \ldots, t_n$ and of $u_1, \ldots, u_n$ to be indistinguishable, not the random variables themselves. As we require function symbols to be interpreted as PPT algorithms, nothing in our semantics can distinguish terms that are equal except for negligible probability and hence = serves as a congruence relation.

Example 3.1. We have introduced a number of function symbols in Example 2.1, which we shall use for analyzing the Diffie-Hellman key exchange protocol. We do not fix the computational implementations of these function symbols, but we assume that whatever the interpretations are, they operate on bit strings, and they satisfy the equations we assumed about them. It is notable that the DDH assumption is for randomly generated groups (group schemes, see Reference [16]), and group generators of those groups. Moreover, the exponents must also be randomly generated. For that reason, the function symbols ggen and r act on names, the interpretation of which are random. We are going to assume that these random groups are such that they satisfy the DDH assumption.

4 PROTOCOLS

The authors of Reference [9] treated protocols as abstract transition systems without committing to any particular way of specifying protocols. They could be specified for instance in the applied pi-calculus [1] or any other process calculus. The authors of Reference [9] also assumed a bounded number of sessions: each protocol comes with an arbitrary but fixed bound on the number of steps in its execution. It would be possible to define the protocols without such a bound, but the general soundness result (Theorem 1 of Reference [9]) holds only for computational adversaries that exploit bounded number of sessions in the security parameter. Therefore, without loss of generality, we can just as well put the bound in the protocol for simplifying the formulation.

4.1 The Transition System

We shall now introduce the abstract transitions systems used in Reference [9]. Observe that in our transition systems, we shall also decorate the states with the names generated in the transition. A protocol is an abstract transition system defined by:

- A finite set of control states Q with a strict partial ordering $>$, an initial state q_0 and a set $Q_f \subseteq Q$ of final states.
- For each state $q \in Q$, a linearly ordered (finite) set $T(q)$ of transition rules

$$q, (N_0, N_1, \ldots, N_n), (\vec{x}) \xrightarrow{\theta} q', (N_0, N_1, \ldots, N_n, N), s, (\vec{x}, x)$$

  $-\vec{x} \equiv x_1, \ldots, x_n$ and x are variables.
  $-N_0, N_1, \ldots, N_n, N$ are lists of names.
  $-\theta$ is a term of sort bool with variables in $x_1, \ldots, x_n, x$
  $-q, q' \in Q$ are such that $q > q'$.
  $-s$ is a term with variables in $x_1, \ldots, x_n, x$.
  $T(q)$ is empty if and only if $q \in Q_f$. Otherwise, $T(q)$ contains a maximal transition, whose guard θ is ***true***.
- An initial knowledge ϕ_0.

Intuitively, a transition $q, (N_0, N_1, \ldots, N_n), (\vec{x}) \xrightarrow{\theta} q', (N_0, N_1, \ldots, N_n, N), s, (\vec{x}, x)$ is a guarded transition that changes the state from q to q' upon receiving the message x; the variables $x_1, \ldots x_n$ store the messages sent by the attacker so far, N_i is the list of names generated upon the receipt of x_i, and the Boolean condition θ specifies the condition under which the transition can be fired (namely, the conditions under which a participating agent moves forward). The term s specifies the message being sent out in the transition, that is, the message sent by the agent with new names in N. The partial ordering on states ensures progress and hence termination. The linear ordering on transitions specifies in which order the guards have to be tried. The ordering on the states thus rules out any non-determinism in the protocol itself.

Example 4.1. The Diffie-Hellman key exchange protocol is the following (see, e.g., Reference [16]):

- A group description G and a group generator element g are generated honestly, according to a randomized algorithm, and made public.
- The Initiator generates a random a in $\mathbb{Z}_{|g|}$ and sends g^a.
- The Responder receives g^a, generates a random b in $\mathbb{Z}_{|g|}$ and sends g^b, and computes $(g^a)^b$.
- The Initiator receives g^b, and computes $(g^b)^a$.

Here, we shall consider two honest parties running two parallel sessions, each of which may be initiator and responder. More sessions can be analyzed similarly: the terms would be much bigger, but there would not be any qualitative difference. As mentioned earlier, the protocol formulation of Reference [9] rules out any non-determinism. The above protocol, however, is not necessarily determinate for the following reason: For example, when agent A has initiated two sessions of the protocol and he receives a response, then it is not clear to which session he will accept the incoming message. For this, we assume that the message coming from the adversary specifies which session the agent should assign it to. Note, the adversary can, of course, direct messages to incorrect sessions thereby creating confusion.

Accordingly, since we want to consider two sessions for each participant, we introduce four session identifiers (message constants in $\mathcal{F}$): two for agent A: $\mathtt{id}_1$, $\mathtt{id}_2$, and two for agent B: $\mathtt{id}_3$ and $\mathtt{id}_4$. We further introduce a function symbol $\mathtt{to} : \mathsf{message} \to \mathsf{message}$, which extracts from an incoming message the session. As for their semantics, the session identifiers can be any fixed, distinct bit strings and to is a function that extracts from a bit string a part that is agreed to be the position for the session identifier. On a bit string that is of the wrong form, the interpretation of to can give an error. We also assume that the session identifiers are distinct: $\mathtt{EQ}(\mathtt{id}_\alpha, \mathtt{id}_\beta) \sim \boldsymbol{false}$ for sessions $\alpha \not\equiv \beta$. Here, we use α, β to denote any of 1, 2, 3, 4. Finally, to ensure that the Initiator also responds something at the end of its role so that execution of other sessions can continue, we introduce an accept message $\mathtt{acc} : \mathsf{message}$.

Then the initiator role of A for session $\mathtt{id}_\alpha$ is the following:

- A receives a message into x_1.
- If $\mathtt{to}(x_1) = \mathtt{id}_\alpha$, and x_1 instructs A to start a new session, then A generates an a in $\mathbb{Z}_{|g|}$, and sends g^a.
- A receives message into x_2.
- If $\mathtt{to}(x_2) = \mathtt{id}_\alpha$, then A computes $\mathtt{m}(x_2)^a$ and sends acc.

The responder role of A is the following:

- A receives a message into y_1.
- If $\mathtt{to}(y_1) = \mathtt{id}_\alpha$, then A generates an a in $\mathbb{Z}_{|g|}$, computes $\mathtt{m}(y_1)^a$, and sends g^a.

We can translate this to a transition system the following way. The set of states Q are given by $\{q^{k_1k_2k_3k_4}_{\ell_1\ell_2\ell_3\ell_4} | k_1, k_2, k_3, k_4 \in \{0, 1, 2\}, \ell_1\ell_2\ell_3\ell_4 \in \{0, 1\}\} \cup \{\bar{q}\}$.

Here, k_α numbers the rounds of session $\mathtt{id}_\alpha$ that has been completed if it is an initiator session, and ℓ_α numbers the round of session $\mathtt{id}_\alpha$ that has been completed if it is a responder session. Clearly, both k_α and l_α cannot be nonzero at the same time for a state. q^{0000}_{0000} is the initial state q_0. The state $\bar{q}$ is the final state where the system jumps if all tests fail. Those $q^{k_1k_2k_3k_4}_{\ell_1\ell_2\ell_3\ell_4}$ states where for each α, either k_α or l_α is maximal are also final.

The transition system is then such that there is a transition corresponding to each pair $(q^{k_1k_2k_3k_4}_{\ell_1\ell_2\ell_3\ell_4}, q^{k_1'k_2'k_3'k_4'}_{\ell_1'\ell_2'\ell_3'\ell_4'})$ where the primed indices are the same as the unprimed ones except for one where the primed is 1 greater than the unprimed. Moreover, for each non-final $q^{k_1k_2k_3k_4}_{\ell_1\ell_2\ell_3\ell_4}$, there is a transition to $\bar{q}$ guarded by ***true*** when all tests fail, which is the last according to the ordering $>$. We do not list all transition rules here as they are rather straightforward but long. We only give some examples:

Consider, for example, the transitions from q^{1002}_{0100}. The only possibilities are to q^{2002}_{0100}, to q^{1012}_{0100}, to q^{1002}_{0110} and to $\bar{q}$. The transitions are with this ordering:

$$q^{1002}_{0100}, (\vec{N}), (\vec{x}) \xrightarrow{\mathsf{EQ}(\mathsf{to}(x_5), \mathtt{id}_1)} q^{2002}_{0100}, (\vec{N}, ()), \mathsf{acc}, (\vec{x}, x_5),$$

$$q^{1002}_{0100}, (\vec{N}), (\vec{x}) \xrightarrow{\mathsf{EQ}(\mathsf{to}(x_5), \mathtt{id}_3)\&\mathsf{EQ}(\mathsf{act}(x_5), \mathsf{new})} q^{1012}_{0100}, (\vec{N}, (n_5)), \mathsf{g}(n_0)^{\mathsf{r}(n_5)}, (\vec{x}, x_5),$$

$$q^{1002}_{0100}, (\vec{N}), (\vec{x}) \xrightarrow{\mathsf{EQ}(\mathsf{to}(x_5), \mathtt{id}_3)} q^{1002}_{0110}, (\vec{N}, (n_5)), \mathsf{g}(n_0)^{\mathsf{r}(n_5)}, (\vec{x}, x_5),$$

$$q^{1002}_{0100}, (\vec{N}), (\vec{x}) \xrightarrow{\textbf{\textit{true}}} \bar{q}, (\vec{N}, ()), \mathbf{0}, (\vec{x}, x_5),$$

where $\vec{x} = x_1, x_2, x_3, x_4$, and $\vec{N} = N_0, N_1, N_2, N_3, N_4$ with each N_α either a fresh name or an empty list. Note here that n_0 is the name used to generate the Diffie-Hellman group description.

Clearly, from q^{0000}_{0000} there are 8 possible transitions by increasing any of the 0's to 1, and there is an additional transition to $\bar{q}$. They are the following as $\alpha = 1, 2, 3, 4$:

$$q^{0000}_{0000}, ((n_0)), () \xrightarrow{\mathsf{EQ}(\mathsf{to}(x_1), \mathtt{id}_\alpha)\&\mathsf{EQ}(\mathsf{act}(x_1), \mathsf{new})} q^{k_\alpha=1}_{0000}, ((n_0), (n_1)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, (x_1),$$

$$q^{0000}_{0000}, ((n_0)), () \xrightarrow{\mathsf{EQ}(\mathsf{to}(x_1), \mathtt{id}_\alpha)} q^{0000}_{\ell_\alpha=1}, ((n_0), (n_1)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, (x_1),$$

$$q^{0000}_{0000}, ((n_0)), () \xrightarrow{\textbf{\textit{true}}} \bar{q}, (n_0, ()), \mathbf{0}, (x_1).$$

That is, if the adversary calls for a new session $\mathtt{id}_\alpha$, then a new initiator session is started. If the adversary sends to sessions $\mathtt{id}_\alpha$ but does not call for a new session, then the agent starts a responder session, and assumes the incoming message is from the initiator.

To make notation more accessible, we present a transition diagram that represents the DH key exchange protocol for two honest agents A and B with two sessions $\mathtt{id}_1$ and $\mathtt{id}_2$ in Figure 1. As expected, a state $q^{k_1,k_2}_{\ell_1,\ell_2}$ will represent the state of each agent A and B. If A (B, respectively) is an initiator, then k_1 (k_2, respectively) will be 1 if waiting for a response and 2, otherwise. If A (B, respectively) is a responder then ℓ_1 (ℓ_2, respectively) will be 1. The diagram illustrates three possible branches of the DH protocol. The right-most branch of q^{00}_{00} simulates the situation where A acts as an initiator and B plays the responder role whereas the left-most branch simulates the other scenario where B plays the initiator role and A plays the responder role. The middle branch illustrates the scenario when all the initiator moves of A happen before the responder moves of B. At the initial state, the honestly generated description of a cyclic group $\mathsf{G}(n)$ and a generator of

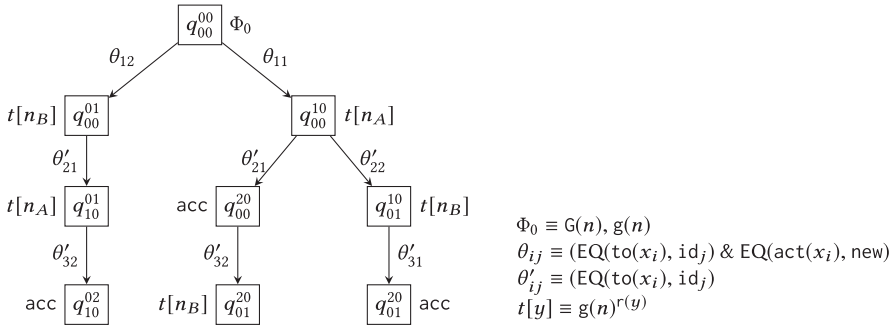


Fig. 1. Diffie-Hellman key exchange protocol with an initiator session $\mathtt{id}_1$ and a responder session $\mathtt{id}_2$.

the group $\mathsf{g}(n)$ are generated honestly and made available to the attacker. This initial knowledge of the attacker is represented by the frame Φ_0. θ_{ij}s represent the conditions that the agent checks if he was instructed to start a new session upon receiving the message x_i in the session $\mathtt{id}_j$ while θ'_{ij}s are the conditions that the agent checks if the message x_i is for the session $\mathtt{id}_j$. The output term $t[y]$ represents the group exponent computed by an agent y in the protocol. Of course, at the end of the protocol, the initiator also sends out the message acc to indicate that the protocol has been completed.

4.2 Execution and Indistinguishability

Computational and symbolic executions were defined precisely in Reference [9]. Instead of repeating the abstract definitions here, we appeal to the reader's intuition, and only illustrate through examples how the executions work symbolically as we carry out the proof there.

We recall first that in case of the symbolic execution, to treat protocol indistinguishability of protocols Π and Π', the Dolev-Yao way would be to match the branches of the execution of Π and that of Π' such that the matched branches are statically equivalent (see, e.g., Reference [19]). However, as the authors discussed in Reference [9], obtaining computational soundness through such matching seems infeasible. Instead, the authors in Reference [9] *folded the protocol* execution into a single trace, such that the tests of the participating agents at each round on the incoming message were included in the terms that were sent out with the help of the function symbol if _ then _ else _ . We illustrate this in the following example.

Example 4.2. The folded symbolic execution of two sessions of the DH protocol between A initiator and B responder has the following trace:

- $\phi_0 \equiv G, g,$
- $\phi_1 \equiv \phi_0, t_1,$

$$
\begin{aligned}
\text{where } t_1 \quad \equiv \quad & \mathsf{if}\ \mathsf{EQ}(\mathsf{to}(f_0(\phi_0)), \mathtt{id}_1)\ \&\ \mathsf{EQ}(\mathsf{act}(f_0(\phi_0)), \mathsf{new}) \\
& \mathsf{then}\ g^{a_1} \\
& \mathsf{else\ if}\ \mathsf{EQ}(\mathsf{to}(f_0(\phi_0)), \mathtt{id}_1) \\
& \qquad \mathsf{then}\ g^{a_1} \\
& \qquad \mathsf{else\ if}\ \mathsf{EQ}(\mathsf{to}(f_0(\phi_0)), \mathtt{id}_2)\ \&\ \mathsf{EQ}(\mathsf{act}(f_0(\phi_0)), \mathsf{new}) \\
& \qquad\qquad \mathsf{then}\ g^{a_2} \\
& \qquad\qquad \mathsf{else\ if}\ \mathsf{EQ}(\mathsf{to}(f_0(\phi_0)), \mathtt{id}_2) \\
& \qquad\qquad\qquad \mathsf{then}\ g^{a_2}
\end{aligned}
$$

$$\begin{array}{l}\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_0(\phi_0)),\mathtt{id}_3)\ \&\ \ldots\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\vdots\\ \qquad\qquad\qquad\qquad\qquad\qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_0(\phi_0)),\mathtt{id}_4)\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{then } g^{a_4}\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{else } \mathbf{0}\end{array}$$

- etc.,

where $G \equiv \pi_1(\mathsf{ggen}(n_0))$ and $g \equiv \pi_2(\mathsf{ggen}(n_0))$ and $a_\alpha \equiv \mathsf{r}(n_{m(\alpha)})$ for $\alpha = 1, 2, 3, 4$ and some increasing function $m : \mathbb{N} \to \mathbb{N}$. Again remember, if the adversary calls for a new session $\mathtt{id}_\alpha$, then a new initiator session is started. If the adversary sends to sessions $\mathtt{id}_\alpha$ but does not call for a new session, then the agent starts a responder session, and assumes the incoming message is from the initiator. To obtain ϕ_2, one proceeds the following way. First, create a term that lists all conditions to reach all possible states after the first step:

$$\begin{array}{l}\text{if } \mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),\mathtt{id}_1)\ \&\ \mathsf{EQ}(\mathtt{act}(f_1(\phi_1)),\mathsf{new})\\ \text{then } q^{1000}_{0000}\\ \text{else if } \mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),\mathtt{id}_1)\\ \qquad\text{then } q^{0000}_{1000}\\ \qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),\mathtt{id}_2)\ \&\ \mathsf{EQ}(\mathtt{act}(f_1(\phi_1)),\mathsf{new})\\ \qquad\qquad\text{then } q^{0100}_{0000}\\ \qquad\qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),\mathtt{id}_2)\\ \qquad\qquad\qquad\text{then } q^{0000}_{0100}\\ \qquad\qquad\qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),\mathtt{id}_3)\ \&\ \ldots\\ \qquad\qquad\qquad\qquad\qquad\vdots\\ \qquad\qquad\qquad\qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),\mathtt{id}_4)\\ \qquad\qquad\qquad\qquad\qquad\text{then } q^{0000}_{0001}\\ \qquad\qquad\qquad\qquad\qquad\text{else } \mathbf{0}\end{array}$$

Then, the states have to be replaced with the terms that describe the transitions out of the state. For example, in Example 4.1, we also listed the transitions from q^{1002}_{0100}. The term that corresponds to these transitions from q^{1002}_{0100} is

$$\begin{array}{l}\text{if } \mathsf{EQ}(\mathtt{to}(f_4(\phi_4)),\mathtt{id}_1)\\ \text{then acc}\\ \text{else if } \mathsf{EQ}(\mathtt{to}(f_4(\phi_4)),\mathtt{id}_3)\ \&\ \mathsf{EQ}(\mathtt{act}(f_4(\phi_4)),\mathsf{new})\\ \qquad\text{then } g^{b_1}\\ \qquad\text{else if } \mathsf{EQ}(\mathtt{to}(f_4(\phi_4)),\mathtt{id}_4)\text{then } g^{b_1}\text{else } \mathbf{0}.\end{array}$$

This way, the indistinguishability of two protocols Π and Π' can be reduced to the indistinguishability of the lists of the sent messages. Let $\mathsf{fold}(\Pi)$ denote the folded execution of protocol Π, and let $\Phi(\mathsf{fold}(\Pi))$ denote the sequence $\phi_0, \phi_1, \ldots$ of folded messages sent on the single symbolic trace. Then the following general soundness theorem was proved in Reference [9]:

Table 1. Core Axioms

Axioms for indistinguishability.	
Refl:	$\vec{x} \sim \vec{x}$
Sym:	$\vec{x} \sim \vec{y} \longrightarrow \vec{y} \sim \vec{x}$
Trans:	$\vec{x} \sim \vec{y} \wedge \vec{y} \sim \vec{z} \longrightarrow \vec{x} \sim \vec{z}$
Restr:	If p projects and permutes onto a sublist, then $\vec{x} \sim \vec{y} \longrightarrow p(\vec{x}) \sim p(\vec{y})$
FuncApp:	for any $\vec{f} : s_1, \ldots, s_n \to s'_1, \ldots, s'_m, \vec{f} \in \mathcal{F} \cup \mathcal{G}, \vec{x} \sim \vec{y} \longrightarrow \vec{x}, \vec{f}(\vec{x}) \sim \vec{y}, \vec{f}(\vec{y})$
TFDist:	$\neg$ ($\boldsymbol{true} \sim \boldsymbol{false}$)
Axioms for equality.	
EqCong:	= is a congruence relation with respect to the current syntax.
EqTheo:	= satisfies the equations given by the equational theory.
Axioms for if _ then _ else _ .	
IfSame:	if b then x else $x \;=\; x$
IfEval:	for any t_1, t_2 terms, if b then $t_1[b]$ else $t_2[b] \;=$ if b then $t_1[\boldsymbol{true}]$ else $t_2[\boldsymbol{false}]$
IfTrue:	if $\boldsymbol{true}$ then x else $y \;= x$
IfFalse:	if $\boldsymbol{false}$ then x else $y \;= y$
IfBranch:	$\vec{z}, b, x_1, \ldots x_n \sim \vec{z}', b', x'_1, \ldots, x'_n \;\wedge\; \vec{z}, b, y_1, \ldots, y_n \sim \vec{z}', b', y'_1, \ldots, y'_n \longrightarrow$ $\vec{z}, b, x_1, \ldots x_n \sim \vec{z}', b', x'_1, \ldots, x'_n \;\wedge\; \vec{z}, b, y_1, \ldots, y_n \sim \vec{z}', b', y'_1, \ldots, y'_n \longrightarrow$ $\vec{z}, b, \begin{smallmatrix}\text{if } b \text{ then } x_1\\ \text{else } y_1\end{smallmatrix}, \ldots, \begin{smallmatrix}\text{if } b \text{ then } x_n\\ \text{else } y_n\end{smallmatrix} \sim \vec{z}', b', \begin{smallmatrix}\text{if } b' \text{ then } x'_1\\ \text{else } y'_1\end{smallmatrix}, \ldots, \begin{smallmatrix}\text{if } b' \text{ then } x'_n\\ \text{else } y'_n\end{smallmatrix}$
Axioms for names.	
FreshInd:	for any names n_1, n_2 and lists of closed terms $\vec{v}, \vec{w}$, such that $\mathsf{fresh}(n_1; \vec{v}, \vec{w})$ and $\mathsf{fresh}(n_2; \vec{v}, \vec{w})$ holds, $\vec{v} \sim \vec{w} \;\longrightarrow\; n_1, \vec{v} \sim n_2, \vec{w}$.
FreshNEq:	for any name n and a closed term v such that $\mathsf{fresh}(n; v)$ holds, we have $\mathsf{EQ}(n, v) \sim \boldsymbol{false}$.

Theorem 4.3. *Let Π, Π' be two protocols. Let $\mathfrak{A}$ be any set of formulas (axioms). If $\mathfrak{A}$ and $\Phi(\mathsf{fold}(\Pi)) \nsim \Phi(\mathsf{fold}(\Pi'))$ are inconsistent, then the protocols Π and Π' are computationally indistinguishable in any computational model $\mathcal{M}^c$ for which $\mathcal{M}^c \models \mathfrak{A}$.*

5 CORE AXIOMS

In this section, we present the core axioms for our technique. In Reference [9] a few axioms were presented that were sufficient to prove the protocol they considered for one session. In general, though, those axioms for if _ then _ else _ are certainly not sufficient to compare the branching of two protocols. In this section, we present further axioms for if _ then _ else _ , and for = as well. As usual, the free variables in axioms are assumed to be universally quantified.

The core axioms are listed in Table 1 and explained below. There are broadly four categories of our axioms. The first category of axioms, *axioms for indistinguishability*, are useful to reason about the indistinguishability predicate $\sim$. The second category of axioms, *axioms for equality*, are useful to reason about the abbreviation =. Collectively, they justify the use of the equality symbol for the abbreviation. The third category of axioms, *axioms for* if _ then _ else lie at the heart of reasoning about different branches of protocol execution. The last category of axioms, *axioms for names*, are useful to reason about fresh names. These axioms (more precisely, axiom schemas) use

the notion of *freshness* [9]: For a list of pairwise distinct names N, and a (possibly empty) list of closed terms $\vec{v}$, $\mathsf{fresh}(N;\vec{v})$ is the constraint that none of names in N occur in $\vec{v}$.

Some of these axioms were proven computationally sound in Reference [9]. The others are proven similarly; we omit their proofs here as they are rather straightforward. Their novelty lies not in the difficulty of their soundness proofs but in their applicability in protocol proofs. The axioms are independent:

Proposition 5.1. *The core axioms are independent.*

The proof goes as usual: For each axiom θ an abstract first-order model is constructed that satisfies all other axioms and the negation of θ.

Note that all axioms we introduce are modular, that is, expanding the logic will not invalidate the current axioms. Observe also the general nature of the axioms; they are in no way special to the DH protocol. They are basic properties that allow us to manipulate if _ then _ else _ branching, equality, and equivalence, and they should be useful in the verification of any protocol. In Section 7, we illustrate their use on simple examples.

Note that the axioms are not necessarily complete. A complete axiomatization would be useful to ensure that a symbolic attack is also a computational attack (realizable with a PPT algorithm). To achieve, however, a complete axiomatization might be very difficult, and it is not an immediate priority from the verification point of view in the sense that soundness is sufficient to ensure that if there is no symbolic attack then there is no computational either. Completeness in restricted cases can be shown though. For example, the following theorem is true (from now on, we use $\vdash$ to denote first-order provability):

Theorem 5.2. *Suppose the only function symbols in t_1 and t_2 are* if _ then _ else _ , ***true*** *and* ***false***. *Then $t_1 = t_2$ is computationally valid, if and only if*
$\mathtt{EqRefl}, \mathtt{EqCong}, \mathtt{IfSame}, \mathtt{IfEval}, \mathtt{IfTrue}, \mathtt{IfFalse} \vdash t_1 = t_2$.

Proof.

(1) Suppose first $t_1 = t_2$ is computationally valid. We prove that the listed axioms imply $t_1 = t_2$ by induction on the number of bool variables in the first arguments of instances of if _ then _ else _ in the formula $t_1 = t_2$.
   (a) Suppose first that there are zero number of such variables. Thanks to `IfTrue`, `IfFalse`, we can assume that t_1 and t_2 have no if _ then _ else _ terms at all. Thus, our formula is $x = y$, where x and y are either variables or ***true*** or ***false***. Clearly, if x and y are syntactically different (that is, $x \not\equiv y$), then $x = y$ is not valid as the variables can just be interpreted as two constant bit strings, different from 1 and 0. When they are syntactically equal, $x = x$, then this is just `EqRefl`.
   (b) Suppose now that we have shown the statement for n different bool variables in $t_1 = t_2$. Consider the case $n + 1$. So either t_1 or t_2 has at least one instance of if _ then _ else _ , suppose w.l.o.g. it is t_1. That means $t_1 =$ if b then t_1^1 else t_1^2 for some b, t_1^1, t_1^2, where by axiom `IfEval`, we can assume that neither t_1^1, nor t_1^2 contains b, and by axioms `IfTrue` and `IfFalse`, we can assume that there is no ***true*** and ***false*** in the first argument of if _ then _ else _ . If t_2 has b in the first argument of if _ then _ else _ , then we can move it out to the front so that $t_2 =$ if b then t_2^1 else t_2^2 for some t_2^1, t_2^2, where we can again assume that neither t_2^1, nor t_2^2 contains b, and ***true*** and ***false*** are removed from the conditions of if _ then _ else _ . If t_2 does not have b, then by `IfSame`, we can still write it as $t_2 =$ if b then t_2^1 else t_2^2 with $t_2^1 = t_2^2 = t_2$. So, we can assume w.l.o.g. that t_2 also

has this form. We claim that $t_1^1 = t_2^1 \ \wedge \ t_1^2 = t_2^2$ is computationally valid. Suppose not, breaking say, $t_1^1 = t_2^1$. Then there is a model $\mathcal{M}^c$ and the variables in $t_1^1, t_1^2, t_2^1, t_2^2$ have an interpretation σ such that $\mathcal{M}^c, \sigma \not\models t_1^1 = t_2^1$. This means that $[\![t_1^1]\!]^\sigma$ and $[\![t_2^1]\!]^\sigma$ are not equal up to negligible probability: $\mathbf{Prob}\{\rho : [\![t_1^1]\!]^\sigma_{\eta,\rho} \neq [\![t_2^1]\!]^\sigma_{\eta,\rho}\}$ is non-negligible. Remember, $t_1^1, t_1^2, t_2^1, t_2^2$ do not contain b. Let us define the interpretation of b (extend σ to b) such that it is a single bit, generated randomly and independently of all the interpretations of other variables in $t_1^1, t_1^2, t_2^1, t_2^2$. By the definition of the semantics of if _ then _ else _ , $\{\rho : [\![b]\!]^\sigma_{\eta,\rho} = 1 \ \wedge \ [\![t_1^1]\!]^\sigma_{\eta,\rho} \neq [\![t_2^1]\!]^\sigma_{\eta,\rho}\} \subseteq \{\rho : [\![t_1]\!]^\sigma_{\eta,\rho} \neq [\![t_2]\!]^\sigma_{\eta,\rho}\}$. Hence, $\frac{1}{2}\mathbf{Prob}\{\rho : [\![t_1^1]\!]^\sigma_{\eta,\rho} \neq [\![t_2^1]\!]^\sigma_{\eta,\rho}\} \leq \mathbf{Prob}\{\rho : [\![t_1]\!]^\sigma_{\eta,\rho} \neq [\![t_2]\!]^\sigma_{\eta,\rho}\}$, and since the left-hand side of this inequality is non-negligible, the right-hand side is also non-negligible. But that means $\mathcal{M}^c, \sigma \models t_1 \neq t_2$, contradicting our assumption. The proof is analogous when $t_1^2 = t_2^2$ is not valid. So, we have that $t_1^1 = t_2^1$ and $t_1^2 = t_2^2$ are both computationally valid. As $t_1^1 = t_2^1$ and $t_1^2 = t_2^2$ do not contain b any more, by the induction hypothesis, both $t_1^1 = t_2^1$ and $t_1^2 = t_2^2$ are derivable from the axioms. Then,

$$t_1 = \text{if } b \text{ then } t_1^1 \text{ else } t_1^2 \quad \overset{\texttt{EqCong}}{=} \quad \text{if } b \text{ then } t_2^1 \text{ else } t_2^2 \quad = \quad t_2.$$

(2) The converse follows immediately from the computational soundness of `EqRefl`, `EqCong`, `IfSame`, `IfEval`, `IfTrue`, `IfFalse`. □

Observe that as an immediate consequence of `EqTheo`, we get:

Example 5.3. For the function symbols in Example 2.1, $\pi_k\langle x_1, x_2\rangle = x_k$ and $g^{ab} = g^{ba}$ are axioms by `EqTheo`.

We could also have defined the axioms differently. The following example indicates that the very intuitive axiom schema `IfEval` and `IfSame` can be replaced by three axioms `IfTF`, `IfIdemp`, `IfMorph` below. Later, we shall use all of `IfSame`, `IfEval`, `IfTF`, `IfIdemp`, `IfMorph`, whichever is more convenient to apply.

Lemma 5.4. *Let us define the following three axioms:*

$$\texttt{IfIdemp}: \quad \mathit{if}\ b\ \mathit{then}\ (\mathit{if}\ b\ \mathit{then}\ x_1\ \mathit{else}\ y_1)\ \mathit{else}\ (\mathit{if}\ b\ \mathit{then}\ x_2\ \mathit{else}\ y_2) = \mathit{if}\ b\ \mathit{then}\ x_1\ \mathit{else}\ y_2,$$

$$\texttt{IfMorph}: \quad f(z_1, \ldots, \mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y, \ldots, z_n) = \begin{array}{l}\mathit{if}\ b\ \ \mathit{then}\ f(z_1, \ldots, x, \ldots, z_n)\\ \qquad\mathit{else}\ f(z_1, \ldots, y, \ldots, z_n),\end{array}$$

$$\texttt{IfTF}: \quad \mathit{if}\ b\ \mathit{then}\ \boldsymbol{\mathit{true}}\ \mathit{else}\ \boldsymbol{\mathit{false}} = b.$$

It is easy to see that `IfTF`, `IfIdemp`, `IfMorph`, `EqRefl`, `EqCong` ⊢ `IfSame`, `IfEval` *and* `IfEval`, `IfSame`, `IfTrue`, `IfFalse`, `EqRefl`, `EqCong` ⊢ `IfTF`, `IfIdemp`, `IfMorph`. *To see the first, we need the transitivity of equality with*

$$\begin{array}{rl}\mathit{if}\ b\ \mathit{then}\ t_1[b]\ \mathit{else}\ t_2[b] & \overset{\texttt{IfTF}}{=} \ \mathit{if}\ b\ \mathit{then}\ t_1[\mathit{if}\ b\ \mathit{then}\ \boldsymbol{\mathit{true}}\ \mathit{else}\ \boldsymbol{\mathit{false}}]\\ & \qquad \mathit{else}\ t_2[\mathit{if}\ b\ \mathit{then}\ \boldsymbol{\mathit{true}}\ \mathit{else}\ \boldsymbol{\mathit{false}}]\\ & \overset{\texttt{IfMorph}}{=} \ \mathit{if}\ b\ \mathit{then}\ \mathit{if}\ b\ \mathit{then}\ t_1[\boldsymbol{\mathit{true}}]\ \mathit{else}\ t_1[\boldsymbol{\mathit{false}}]\\ & \qquad \mathit{else}\ \mathit{if}\ b\ \mathit{then}\ t_2[\boldsymbol{\mathit{true}}]\ \mathit{else}\ t_2[\boldsymbol{\mathit{false}}]\\ & \overset{\texttt{IfIdemp}}{=} \ \mathit{if}\ b\ \mathit{then}\ t_1[\boldsymbol{\mathit{true}}]\ \mathit{else}\ t_2[\boldsymbol{\mathit{false}}]\end{array}$$

and[1]

$$x \overset{\texttt{IfTrue}}{=} \mathit{if}\ \boldsymbol{true}\ \mathit{then}\ x\ \mathit{else}\ \mathit{if}\ b\ \mathit{then}\ y\ \mathit{else}\ z \overset{\texttt{IfMorph}}{=} \mathit{if}\ b\ \mathit{then}\ (\mathit{if}\ \boldsymbol{true}\ \mathit{then}\ x\ \mathit{else}\ y\ )\ \mathit{else}\ (\mathit{if}\ \boldsymbol{true}\ \mathit{then}\ x\ \mathit{else}\ z\ ) \overset{\texttt{IfTrue}}{=} \mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ x$$

To see the second,

$$b \overset{\texttt{IfSame}}{=} \mathit{if}\ b\ \mathit{then}\ b\ \mathit{else}\ b \overset{\texttt{IfEval}}{=} \mathit{if}\ b\ \mathit{then}\ \boldsymbol{true}\ \mathit{else}\ \boldsymbol{false}$$

$$\begin{aligned} f(z_1,\ldots,\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y\ ,\ldots,z_n) &\overset{\texttt{IfSame}}{=} \mathit{if}\ b\ \mathit{then}\ f(z_1,\ldots,\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y\ ,\ldots,z_n) \\ &\qquad \mathit{else}\ f(z_1,\ldots,\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y\ ,\ldots,z_n) \\ &\overset{\texttt{IfEval}}{=} \mathit{if}\ b\ \mathit{then}\ f(z_1,\ldots,\mathit{if}\ \boldsymbol{true}\ \mathit{then}\ x\ \mathit{else}\ y\ ,\ldots,z_n) \\ &\qquad \mathit{else}\ f(z_1,\ldots,\mathit{if}\ \boldsymbol{false}\ \mathit{then}\ x\ \mathit{else}\ y\ ,\ldots,z_n) \\ &\overset{\substack{\texttt{IfTrue}\\ \texttt{IfFalse}}}{=} \mathit{if}\ b\ \mathit{then}\ f(z_1,\ldots,x,\ldots,z_n)\ \mathit{else}\ f(z_1,\ldots,y,\ldots,z_n) \end{aligned}$$

$$\begin{aligned} \mathit{if}\ b\ \mathit{then}\begin{pmatrix}\mathit{if}\ b\ \mathit{then}\ x_1\\ \mathit{else}\ y_1\end{pmatrix}\ \mathit{else}\begin{pmatrix}\mathit{if}\ b\ \mathit{then}\ x_2\\ \mathit{else}\ y_2\end{pmatrix} &\overset{\texttt{IfEval}}{=} \mathit{if}\ b\ \mathit{then}\begin{pmatrix}\mathit{if}\ \boldsymbol{true}\ \mathit{then}\ x_1\\ \mathit{else}\ y_1\end{pmatrix} \\ &\qquad \mathit{else}\begin{pmatrix}\mathit{if}\ \boldsymbol{false}\ \mathit{then}\ x_2\\ \mathit{else}\ y_2\end{pmatrix} \\ &\overset{\substack{\texttt{IfTrue}\\ \texttt{IfFalse}}}{=} \mathit{if}\ b\ \mathit{then}\ x_1\ \mathit{else}\ y_2\ . \end{aligned}$$

5.1 Soundness of the Axioms

The soundness of the axioms for indistinguishability were proven in Reference [9] except for TFDist. But that is trivial: the interpretation of ***true*** is identically 1, the interpretation of ***false*** is identically 0, which can be distinguished by the algorithm that outputs its input.

Proposition 5.5. *Axioms* EqRefl, EqCong, *and* EqTheo *are computationally sound.*

Proof. $x = x$ is trivial by the semantics of EQ and ***true***.

To see EqCong, note that by the definition of the semantics of EQ and ***true***, $\mathcal{M}^c \models \mathsf{EQ}(x, y) \sim \boldsymbol{true}$ means that $[\![x]\!]$ and $[\![y]\!]$ are equal on all inputs except possibly some inputs that have negligible probability. As any change that affects the outputs only with negligible probability does not affect the satisfaction of formulas expressed by the current syntax ($\sim$ ignores any change with negligible probability), congruence indeed holds.

Finally, for EqTheo, if the computational semantics satisfies the equations bitwise, then for any given equation (of the equational theory), the interpretations of the two terms on the two sides agree on each input. Hence, they are equal up to negligible probability as well. □

Proposition 5.6. *The* If *axioms are computationally sound.*

Proof. IfSame, IfEval, IfTrue, IfFalse: These axioms are all of the form $t_1 = t_2$ with t_1 and t_2 terms varying from axiom to axiom. Assume that σ is an assignment of the variables of t_1 and t_2 to algorithms taking values in the corresponding domains. Let η be a security parameter. In each case, by the definition of $[\![\text{if \_ then \_ else \_}]\!]$, it is a trivial matter to verify that $[\![t_1]\!]^{\sigma}_{\eta,\rho} = [\![t_2]\!]^{\sigma}_{\eta,\rho}$.

[1] This observation is due to Adrien Koutsos.

Then, by the definition of $[\![\mathtt{EQ}(\_,\_)]\!]$ and $[\![\boldsymbol{true}]\!]$, we have that $\mathtt{EQ}(t_1,t_2) \sim \boldsymbol{true}$ is satisfied and that completes the proof.

$$\mathtt{IfBranch}\colon \vec{z}, b, x_1, \ldots x_n \sim \vec{z}', b', x'_1, \ldots, x'_n \ \wedge\ \vec{z}, b, y_1, \ldots, y_n \sim \vec{z}', b', y'_1, \ldots, y'_n \longrightarrow$$

$$\vec{z}, b, \begin{array}{c}\text{if } b \text{ then } x_1\\ \text{else } y_1\end{array}, \ldots, \begin{array}{c}\text{if } b \text{ then } x_n\\ \text{else } y_n\end{array} \sim \vec{z}', b', \begin{array}{c}\text{if } b' \text{ then } x'_1\\ \text{else } y'_1\end{array}, \ldots, \begin{array}{c}\text{if } b' \text{ then } x'_n\\ \text{else } y'_n\end{array}$$

Assume an assignment σ of the free variables $\vec{z}, \vec{z}', b, b', x_1, \ldots, x_n, x'_1, \ldots, x'_n, y_1, \ldots, y_n, y'_1, \ldots, y'_n$, taking values in the corresponding domains and a security parameter η. Assume further that $\vec{z}, b, x_1, \ldots x_n \sim \vec{z}', b', x'_1, \ldots, x'_n$ and $\vec{z}, b, y_1, \ldots, y_n \sim \vec{z}', b', y'_1, \ldots, y'_n$. Fix an adversary $\mathcal{A}$. Let $p_l, p_r, p_x, p_{x'}, p_y$, and $p_{y'}$ be defined as follows:

$$\begin{aligned}
p_l &= \mathbf{Prob}\{\rho\colon \mathcal{A}([\![\vec{z}, b, \text{if } b \text{ then } x_1 \text{ else } y_1, \ldots, \text{if } b \text{ then } x_n \text{ else } y_n\ ]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\},\\
p_r &= \mathbf{Prob}\{\rho\colon \mathcal{A}([\![\vec{z}', b', \text{if } b' \text{ then } x'_1 \text{ else } y'_1, \ldots, \text{if } b' \text{ then } x'_n \text{ else } y'_n\ ]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\},\\
p_x &= \mathbf{Prob}\{\rho:\ \mathcal{A}([\![\vec{z}, b, x_1, \ldots, x_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\ \&\ [\![b]\!]^{\sigma}_{\rho,\eta} = 1\},\\
p_{x'} &= \mathbf{Prob}\{\rho:\ \mathcal{A}([\![\vec{z}', b', x'_1, \ldots, x'_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\ \&\ [\![b']\!]^{\sigma}_{\rho,\eta} = 1\},\\
p_y &= \mathbf{Prob}\{\rho:\ \mathcal{A}([\![\vec{z}, b, y_1, \ldots, y_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\ \&\ [\![b]\!]^{\sigma}_{\rho,\eta} = 0\},\\
p_{y'} &= \mathbf{Prob}\{\rho:\ \mathcal{A}([\![\vec{z}', b', y'_1, \ldots, y'_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\ \&\ [\![b']\!]^{\sigma}_{\rho,\eta} = 0\}.
\end{aligned}$$

It is easy to see that $p_l = p_x + p_y$ and that $p_r = p_{x'} + p_{y'}$. Therefore, $|p_l - p_r| \leq |p_x - p_{x'}| + |p_y - p_{y'}|$. To prove the soundness of the axiom, we need to show that $|p_l - p_r|$ is negligible in η. To prove this, it suffices to show that both $|p_x - p_{x'}|$ are $|p_y - p_{y'}|$ are negligible in η.

We now show that $|p_x - p_{x'}|$ is negligible in η. Let the sequence $\vec{m}^1$ have the same number of elements as $\vec{z}$. Consider the adversary $\mathcal{B}$ that on input $\vec{m}^1, b_1, m_1, \ldots, m_n$ and random tape ρ_2 runs $\mathcal{A}(\vec{m}^1, b_1, m_1, \ldots, m_n)$ when b_1 is 1 and outputs 0, otherwise. Now, it is easy to see that

$$\mathbf{Prob}\{\rho:\ \mathcal{B}([\![\vec{z}, b, x_1, \ldots, x_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\} = p_x$$

and

$$\mathbf{Prob}\{\rho:\ \mathcal{B}([\![\vec{z}', b', x'_1, \ldots, x'_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\} = p_{x'}.$$

Thus,

$$\begin{aligned}
|p_x - p_{x'}| = |&\mathbf{Prob}\{\rho:\ \mathcal{B}([\![\vec{z}, b, x_1, \ldots, x_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\} -\\
&\mathbf{Prob}\{\rho:\ \mathcal{B}([\![\vec{z}', b', x'_1, \ldots, x'_n]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\}|.
\end{aligned}$$

Now, the latter is negligible in η as $\vec{z}, b, x_1, \ldots, x_n \sim \vec{z}', b', x'_1, \ldots, x'_n$. Hence, $|p_x - p_{x'}|$ is also negligible in η. Similarly, we can show that $|p_y - p_{y'}|$ is negligible in η and the result follows. □

Proposition 5.7. *Axioms* `FreshInd` *and* `FreshNEq` *are computationally sound.*

Proof. For `FreshInd`, note that $\mathtt{fresh}(n_1, n_2; \vec{v}, \vec{w})$ implies that $[\![n_1]\!]$ and $[\![n_2]\!]$ are independent of $[\![\vec{v}, \vec{w}]\!]$, because all names are assumed to use different parts of the random tape ρ_1, and functions can only use randomness from ρ_2. This means that $[\![n_2, \vec{w}]\!]$ and $[\![n_1, \vec{w}]\!]$ have identical probability distributions. Hence, if an algorithm $\mathcal{A}$ can differentiate $[\![n_1, \vec{v}]\!]$ from $[\![n_2, \vec{w}]\!]$, then $\mathcal{A}$ can also differentiate $[\![n_1, \vec{v}]\!]$ from $[\![n_1, \vec{w}]\!]$. If there is such an $\mathcal{A}$, then there is also a $\mathcal{B}$ differentiating $[\![\vec{v}]\!]$ and $[\![\vec{w}]\!]$, namely, the one that generates a random bit string s that has identical distribution with the interpretation of names, and then gives $(s, [\![\vec{v}]\!])$ or $(s, [\![\vec{w}]\!])$ to $\mathcal{A}$.

To see soundness of `FreshNEq`, note again that $[\![n]\!]$ and $[\![v]\!]$ are independent. As $[\![n]\!]$ has uniform distribution on $\{0,1\}^{p(\eta)}$, there is at most negligible probability for $[\![n]\!]$ to agree with $[\![v]\!]$,

and hence there is only negligible probability for $[\![\mathtt{EQ}(n, v)]\!]$ to be nonzero, from which soundness follows. □

6 DDH ASSUMPTION

The BC formalism for indistinguishability properties is very convenient for axiomatizing cryptographic assumptions. Our Decisional Diffie-Hellman (DDH) axiom is a straightforward translation of the usual DDH assumption to this formalism:

- DDH assumption:
  $\mathsf{fresh}(n, n_1, n_2, n_3) \longrightarrow (\mathtt{G}(n), \mathtt{g}(n), \mathtt{g}(n)^{\mathtt{r}(n_1)}, \mathtt{g}(n)^{\mathtt{r}(n_2)}, \mathtt{g}(n)^{\mathtt{r}(n_1)\mathtt{r}(n_2)}) \sim (\mathtt{G}(n), \mathtt{g}(n), \mathtt{g}(n)^{\mathtt{r}(n_1)}, \mathtt{g}(n)^{\mathtt{r}(n_2)}, \mathtt{g}(n)^{\mathtt{r}(n_3)})$.

That is, this property postulates that an adversary cannot distinguish $\mathtt{g}(n)^{\mathtt{r}(n_1)\mathtt{r}(n_2)}$ from $\mathtt{g}(n)^{\mathtt{r}(n_3)}$ even if the items $\mathtt{G}(n)$, $\mathtt{g}(n)$, $\mathtt{g}(n)^{\mathtt{r}(n_1)}$, $\mathtt{g}(n)^{\mathtt{r}(n_2)}$ are disclosed.

Proposition 6.1. *The above axiom is sound if and only if the interpretation of* $(\mathtt{ggen}(\_), \mathtt{r}(\_))$ *satisfies the Decisional Diffie-Hellman assumption (see, for example, Reference [16]).*

Proof. The proof is almost trivial. According to the semantics of $\sim$ in Section 3.2, violation of the DDH axiom means there is an $\mathcal{A}$ algorithm for which the advantage is non-negligible when it is fed with the interpretation of $(\mathtt{G}(n), \mathtt{g}(n), \mathtt{g}(n)^{\mathtt{r}(n_1)}, \mathtt{g}(n)^{\mathtt{r}(n_2)}, \mathtt{g}(n)^{\mathtt{r}(n_1)\mathtt{r}(n_2)})$ and the interpretation of $(\mathtt{G}(n), \mathtt{g}(n), \mathtt{g}(n)^{\mathtt{r}(n_1)}, \mathtt{g}(n)^{\mathtt{r}(n_2)}, \mathtt{g}(n)^{\mathtt{r}(n_3)})$. That is exactly the violation of the DDH assumption in Reference [16], Definition 2.1. □

7 SHORT EXAMPLES

In this section, we illustrate with a few short examples how the axioms we introduced work.

Example 7.1. In the formula below, `IfMorph` lets us pull out if _ then _ else _ from under t_1, t_2, and `IfIdemp` lets us get rid of several instances of b. And, as `EqRefl` and `EqCong` imply transitivity of =, we have

$$\mathtt{IfIdemp}, \mathtt{IfMorph}, \mathtt{EqRefl}, \mathtt{EqCong} \vdash \begin{array}{r}\mathsf{if}\ b\ \mathsf{then}\ t_1[\mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ y_1\ ] \\ \mathsf{else}\ t_2[\mathsf{if}\ b\ \mathsf{then}\ x_2\ \mathsf{else}\ y_2\ ]\end{array} = \begin{array}{r}\mathsf{if}\ b\ \mathsf{then}\ t_1[x_1] \\ \mathsf{else}\ t_2[y_2].\end{array}$$

Example 7.2. We have that for any constant $f \in \mathcal{F} \cup \mathcal{G}$,

$$\mathtt{Trans}, \mathtt{Restr}, \mathtt{FuncApp}, \mathtt{EqRefl} \vdash x \sim f \longrightarrow x = f.$$

To see this, consider $x \sim f$. By `FuncApp`, $x, f \sim f, f$, and again by `FuncApp`, $x, f, \mathtt{EQ}(x, f) \sim f, f, \mathtt{EQ}(f, f)$. By `Restr`, $\mathtt{EQ}(x, f) \sim \mathtt{EQ}(f, f)$. By `EqRefl`, $f = f$, which is a shorthand for $\mathtt{EQ}(f, f) \sim \boldsymbol{true}$. Then by `Trans`, $\mathtt{EQ}(x, f) \sim \boldsymbol{true}$, which is $x = f$. Note that in particular, $x = y$ iff $\mathtt{EQ}(x, y) = \boldsymbol{true}$.

Example 7.3. We have

$$\mathtt{Trans}, \mathtt{Restr}, \mathtt{FuncApp}, \mathtt{IfSame}, \mathtt{IfIdemp}, \mathtt{IfMorph}, \mathtt{IfTF}, \mathtt{EqRefl}, \mathtt{EqCong} \vdash$$
$$\mathsf{if}\ \mathtt{EQ}(x_1, x_2)\ \mathsf{then}\ x_1\ \mathsf{else}\ y\ = \mathsf{if}\ \mathtt{EQ}(x_1, x_2)\ \mathsf{then}\ x_2\ \mathsf{else}\ y.$$

This is because

$$\begin{aligned}
\mathsf{EQ}\begin{pmatrix}\text{if } \mathsf{EQ}(x_1,x_2) \text{ then } x_1 \text{ else } y\,,\\ \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } x_2 \text{ else } y\end{pmatrix} &\overset{\texttt{IfMorph}}{=} \begin{array}{l}\text{if } \mathsf{EQ}(x_1,x_2)\\ \text{then } \mathsf{EQ}\begin{pmatrix}x_1,\\ \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } x_2 \text{ else } y\end{pmatrix}\\ \text{else } \mathsf{EQ}\begin{pmatrix}y,\\ \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } x_2 \text{ else } y\end{pmatrix}\end{array}\\
&\overset{\text{Example 7.1}}{=} \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } \mathsf{EQ}(x_1,x_2) \text{ else } \mathsf{EQ}(y,y)\\
&\overset{\text{Example 7.2}}{=} \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } \mathsf{EQ}(x_1,x_2) \text{ else } \boldsymbol{true}\\
&\overset{\texttt{IfEval}}{=} \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{true} \overset{\texttt{IfSame}}{=} \boldsymbol{true},
\end{aligned}$$

where EqCong is also used, but we omitted its indication.

Example 7.4. We prove the following:

$$\texttt{IfIdemp}, \texttt{IfMorph}, \texttt{EqRefl}, \texttt{EqCong} \vdash \text{if } b \text{ then } x_1 \text{ else } y_1 = \text{if } b \text{ then } x_2 \text{ else } y_2 \longrightarrow \text{if } b \text{ then } t[x_1] \text{ else } t'[y_1] = \text{if } b \text{ then } t[x_2] \text{ else } t'[y_2].$$

The statement can be proven using Example 7.1 and congruence of the equality:

$$\begin{aligned}
\text{if } b \text{ then } t[x_1] \text{ else } t'[y_1] &= \text{if } b \text{ then } t[\text{if } b \text{ then } x_1 \text{ else } y_1\ ] \text{ else } t'[\text{if } b \text{ then } x_1 \text{ else } y_1\ ]\\
&= \text{if } b \text{ then } t[\text{if } b \text{ then } x_2 \text{ else } y_2\ ] \text{ else } t'[\text{if } b \text{ then } x_2 \text{ else } y_2\ ]\\
&= \text{if } b \text{ then } t[x_2] \text{ else } t'[y_2].
\end{aligned}$$

Putting this together with Example 7.3, we have in particular that[2]

$$\texttt{EqBranch} \equiv \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } t[x_1] \text{ else } t' = \text{if } \mathsf{EQ}(x_1,x_2) \text{ then } t[x_2] \text{ else } t'.$$

These two previous examples mean that equality is not only a congruence, but if x_1 and x_2 are equal on a branch, they are interchangeable on that particular branch.

Example 7.5. By the previous examples, we have $\mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false}) = \boldsymbol{false}$ as follows:

$$\begin{aligned}
\mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false}) &\overset{\texttt{IfTF}}{=} \text{if } \mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false}) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false}\\
&\overset{\texttt{EqBranch}}{=} \text{if } \mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false}) \text{ then } \boldsymbol{false} \text{ else } \boldsymbol{false} \overset{\texttt{IfSame}}{=} \boldsymbol{false}.
\end{aligned}$$

Example 7.6. By the previous examples, we also have $\mathsf{EQ}(x,y) = \mathsf{EQ}(y,x)$.
The proof is the following:

$$\begin{aligned}
\mathsf{EQ}(x,y) &\overset{\texttt{IfTF}}{=} \text{if } \mathsf{EQ}(x,y) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false}\\
&\overset{\texttt{IfEval}}{=} \text{if } \mathsf{EQ}(x,y) \text{ then } \mathsf{EQ}(x,y) \text{ else } \boldsymbol{false}\\
&\overset{\texttt{EqBranch}}{=} \text{if } \mathsf{EQ}(x,y) \text{ then } \mathsf{EQ}(y,x) \text{ else } \boldsymbol{false}\\
&\overset{\texttt{IfTF}}{=} \text{if } \mathsf{EQ}(x,y) \text{ then } (\text{if } \mathsf{EQ}(y,x) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false}\ ) \text{ else } \boldsymbol{false}\\
&\overset{\texttt{IfMorph}}{=} \begin{array}{l}\text{if } \mathsf{EQ}(y,x) \text{ then } (\text{if } \mathsf{EQ}(x,y) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false}\ )\\ \quad\text{else } (\text{if } \mathsf{EQ}(x,y) \text{ then } \boldsymbol{false} \text{ else } \boldsymbol{false}\ )\end{array}
\end{aligned}$$

[2]This property was first formulated by Adrien Koutsos as one that is particularly useful in proofs.

$$\overset{\texttt{IfSame}}{=} \text{ if } \mathsf{EQ}(y,x) \text{ then (if } \mathsf{EQ}(x,y) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false}\text{) else } \boldsymbol{false}$$

$$\overset{\texttt{IfTF}}{=} \text{ if } \mathsf{EQ}(y,x) \text{ then } \mathsf{EQ}(x,y) \text{ else } \boldsymbol{false}$$

$$\overset{\texttt{EqBranch}}{=} \text{ if } \mathsf{EQ}(y,x) \text{ then } \mathsf{EQ}(y,x) \text{ else } \boldsymbol{false}$$

$$\overset{\texttt{IfEval}}{=} \text{ if } \mathsf{EQ}(y,x) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false} \overset{\texttt{IfTF}}{=} \mathsf{EQ}(y,x)$$

Example 7.7. By the invertibility of pairing, we can also show that for two distinct names n_1 and n_2, $\mathsf{EQ}(n_1, \langle n_1, n_2\rangle) = \boldsymbol{false}$.

To see this, note that from the equational theory of the pairing, $\pi_2(\langle n_1, n_2\rangle) = n_2$, which, by Axiom EqTheo, the meaning of = as an abbreviation, and Example 7.2, means $\mathsf{EQ}(n_2, \pi_2(\langle n_1, n_2\rangle)) = \boldsymbol{true}$. Then,

$$\begin{aligned}
\mathsf{EQ}(n_1, \langle n_1, n_2\rangle) &\overset{\texttt{IfTF}}{=} \text{ if } \mathsf{EQ}(n_1, \langle n_1, n_2\rangle) \text{ then } \boldsymbol{true} \text{ else } \boldsymbol{false} \\
&\overset{\substack{\texttt{EqTheo}\\ \text{Example 7.2}}}{=} \text{ if } \mathsf{EQ}(n_1, \langle n_1, n_2\rangle) \text{ then } \mathsf{EQ}(n_2, \pi_2(\langle n_1, n_2\rangle)) \text{ else } \boldsymbol{false} \\
&\overset{\texttt{EqBranch}}{=} \text{ if } \mathsf{EQ}(n_1, \langle n_1, n_2\rangle) \text{ then } \mathsf{EQ}(n_2, \pi_2(n_1)) \text{ else } \boldsymbol{false} \\
&\overset{\substack{\texttt{FreshNEq}\\ \text{Example 7.2}}}{=} \text{ if } \mathsf{EQ}(n_1, \langle n_1, n_2\rangle) \text{ then } \boldsymbol{false} \text{ else } \boldsymbol{false} \overset{\texttt{IfSame}}{=} \boldsymbol{false}
\end{aligned}$$

Example 7.8. It is easy to see from the definition of not that

$$\texttt{EqRefl}, \texttt{EqCong}, \texttt{IfMorph}, \texttt{IfTrue}, \texttt{IfFalse} \vdash \text{if } \mathsf{not}(b) \text{ then } x \text{ else } y \;=\; \text{if } b \text{ then } y \text{ else } x.$$

Example 7.9. Axioms IfIdemp and IfMorph reduce terms in the frame significantly for the following reason. Consider the simple situation when the frame ϕ_2 is defined as follows:

$$\phi_1 \equiv \phi_0, \text{if } b_1[f_1(\phi_0)] \text{ then } t_1^1[f_1(\phi_0)] \text{ else } t_1^2[f_1(\phi_0)],$$

$$\phi_2 \equiv \phi_1, \begin{array}{r} \text{if } b_1[f_1(\phi_0)] \text{ then } \left(\text{if } b_2^1[f_2(\phi_1)] \text{ then } t_2^{11}[f_2(\phi_1)] \text{ else } t_2^{12}[f_2(\phi_1)]\right) \\ \text{else } \left(\text{if } b_2^2[f_2(\phi_1)] \text{ then } t_2^{01}[f_2(\phi_1)] \text{ else } t_2^{00}[f_2(\phi_1)]\right). \end{array}$$

Inside f_2, the ϕ_1 also has branching, but by axiom IfIdemp and IfMorph, that branching can be removed. So the last term in ϕ_2 is equal to

$$\begin{array}{r} \text{if } b_1[f_1(\phi_0)] \text{ then } \left(\text{if } b_2^1[f_2(\phi_0, t_1^1[f_1(\phi_0)])] \text{ then } t_2^{11}[f_2(\phi_0, t_1^1[f_1(\phi_0)])] \text{ else } t_2^{12}[f_2(\phi_0, t_1^1[f_1(\phi_0)])]\right) \\ \text{else } \left(\text{if } b_2^2[f_2(\phi_0, t_1^2[f_1(\phi_0)])] \text{ then } t_2^{01}[f_2(\phi_0, t_1^2[f_1(\phi_0)])] \text{ else } t_2^{00}[f_2(\phi_0, t_1^2[f_1(\phi_0)])]\right). \end{array}$$

Similarly, even in later terms of the frame, all the branching in the adversary messages (as in the t's above) can be removed. Note that because of the way terms were folded in the protocol execution, the branching is always kept as we go to higher elements of the frame, they only get extended: Just as above, there is an initial branching by b_1, then there is a second by b_2, then b_3, and they all show up in all later terms as well. That is, ϕ_{n+1} will have the same branching as ϕ_n plus an additional layer of branching, and these branchings can all be pulled out to the front of the terms.

Example 7.10. In this example, we show that a three-party version of the DDH assumption can be derived from the usual DDH assumption. In this case, $G, g, g^a, g^b, g^c, g^{ab}, g^{ac}, g^{bc}$, are all public,

and g^{abc} is secret. We show that with all this public information, g^{abc} is indistinguishable from g^e where e is a freshly generated exponent. More precisely, we show the following:

$$\mathsf{fresh}(G,g,a,b,c,e) \rightarrow \begin{pmatrix} G,g, & g^a, & g^b, & g^c, \\ g^{ab}, & g^{ac}, & g^{bc}, & g^{abc} \end{pmatrix} \sim \begin{pmatrix} G,g, & g^a, & g^b, & g^c, \\ g^{ab}, & g^{ac}, & g^{bc}, & g^{e} \end{pmatrix}$$

(1) Take a d with $\mathsf{fresh}(G,g,a,b,c,e,d)$.
(2) Thanks to DDH axiom, we have that $G,g,g^a,g^b,g^{ab} \sim G,g,g^a,g^b,g^d$.
(3) From line 2 and axiom FreshInd, we get that $c,G,g,g^a,g^b,g^{ab} \sim c,G,g,g^a,g^b,g^d$.
(4) From line 3 and repeated use of axiom FuncApp (for exponentiation with c), we get that $c,G,g,g^a,g^b,g^{ab},g^c,g^{ac},g^{bc},g^{abc} \sim c,G,g,g^a,g^b,g^d,g^c,g^{ac},g^{bc},g^{dc}$
(5) From line 4 and axiom Restr, we get that $G,g,g^a,g^b,g^{ab},g^c,g^{ac},g^{bc},g^{abc} \sim G,g,g^a,g^b,g^d,g^c,g^{ac},g^{bc},g^{dc}$
(6) By DDH, we get that $G,g,g^d,g^c,g^{dc} \sim G,g,g^d,g^c,g^e$
(7) From line 6 and axiom FreshInd, we get that $a,b,G,g,g^d,g^c,g^{dc} \sim a,b,G,g,g^d,g^c,g^e$
(8) From line 7 and repeated use of axiom FuncApp, we get that $a,b,G,g,g^d,g^c,g^{dc}g^a,g^b,g^{ca},g^{cb} \sim a,b,G,g,g^d,g^c,g^eg^a,g^b,g^{ca},g^{cb}$
(9) Since we have postulated that $g^{ca}=g^{ac}$ and that $g^{cb}=g^{bc}$, we get thanks to Line 8 and axiom Restr that $G,g,g^a,g^b,g^d,g^c,g^{ac},g^{bc},g^{dc} \sim G,g,g^a,g^b,g^d,g^c,g^{ac},g^{bc},g^e$
(10) Thanks to axiom Trans, lines 5 and 9, we get that $G,g,g^a,g^b,g^{ab},g^c,g^{ac},g^{bc},g^{abc} \sim G,g,g^a,g^b,g^d,g^c,g^{ac},g^{bc},g^e$
(11) Now, thanks to line 2 and axiom FreshInd, $c,e,G,g,g^a,g^b,g^{ab} \sim c,e,G,g,g^a,g^b,g^d$
(12) Thanks to line 11 and repeated use of axiom FuncApp, we get that $c,e,G,g,g^a,g^b,g^{ab},g^c,g^e,g^{ac},g^{bc} \sim c,e,G,g,g^a,g^b,g^d,g^c,g^e,g^{ac},g^{bc}$
(13) Thanks to line 12, axiom Restr and Sym, we get that $G,g,g^a,g^b,g^d,g^c,g^{ac},g^{bc},g^e \sim G,g,g^a,g^b,g^{ab},g^c,g^{ac},g^{bc},g^e$
(14) Now, thanks to lines 10, 13 and axiom Trans, we get that $G,g,g^a,g^b,g^{ab},g^c,g^{ac},g^{bc},g^{abc} \sim G,g,g^a,g^b,g^{ab},g^c,g^{ac},g^{bc},g^e$
The result follows.

Example 7.11. The Diffie-Hellman assumption does not imply that for $\mathsf{fresh}(G,g,a,b,c,d,e)$, the equivalence $G,g,g^a,g^b,g^c,g^{ab},g^{bc} \sim G,g,g^a,g^b,g^c,g^d,g^e$ holds. However, we do have that for β closed bool term on g,g^a,g^b,g^c and function symbols, if $\mathsf{fresh}(G,g,a,b,c,d)$ holds, then

$$G,g,g^a,g^b,g^c,\mathsf{if}\ \beta\ \mathsf{then}\ g^{ab}\ \mathsf{else}\ g^{bc} \sim G,g,g^a,g^b,g^c,g^d.$$

This can be derived the following way: from the DDH assumption, $G,g,g^a,g^b,g^{ab} \sim G,g,g^a,g^b,g^d$. By FreshInd, we also have that $c,G,g,g^a,g^b,g^{ab} \sim c,G,g,g^a,g^b,g^d$, and by FuncApp and Restr, $G,g,g^a,g^b,g^c,g^{ab} \sim G,g,g^a,g^b,g^c,g^d$. As β is a closed term on g,g^a,g^b,g^c, by FuncApp and Restr again, we have $G,g,g^a,g^b,g^c,\beta,g^{ab} \sim G,g,g^a,g^b,g^c,\beta,g^d$. Similarly, $G,g,g^a,g^b,g^c,\beta,g^{bc} \sim G,g,g^a,g^b,g^c,\beta,g^d$. Then, by IfBranch, we obtain $G,g,g^a,g^b,g^c,\mathsf{if}\ \beta\ \mathsf{then}\ g^{ab}\ \mathsf{else}\ g^{bc} \sim G,g,g^a,g^b,g^c,\mathsf{if}\ \beta\ \mathsf{then}\ g^d\ \mathsf{else}\ g^d$, and finally by IfSame, we get what we wanted to prove.

8 DIFFIE-HELLMAN KEY EXCHANGE

Let us come back now to our running example of the Diffie-Hellman key exchange protocol. In this section, we show that if the group scheme used for the key exchange protocol satisfies the DDH assumption, then the shared key satisfies real-or-random secrecy. More precisely, we show that two protocols, one in which the real shared key g^{ab} is published at the end and one in which

g^d is published with a freshly generated d, are indistinguishable. Real-or-random secrecy was introduced in Reference [3]. According to their definition, the adversary can request an oracle to reveal the shared key of the honest agents. The oracle either reveals the true shared key, or it reveals a newly generated random key, and the adversary has to guess whether the real or the freshly generated random key was revealed. Real-or-random secrecy holds if the attacker guesses correctly with a probability at most negligibly exceeding 1/2.

Note, the basic DH protocol does not ensure authentication: agents A and B have no way to know if they really communicate with each other. For example, if the adversary sends some bit string s to A, the key that A generates, s^a will not be secret. Accordingly, the oracle has to choose those keys between A and B that were indeed honestly computed and shared. Only those keys have a chance to remain secret. Hence, the oracle takes a session (specifying the agent as well) as an input and checks if there is a matching session. If there is no matching session, then it outputs the key computed by the agent. If, however, there is a matching session, then the oracle outputs either the real key, or generates a new g^c and outputs that. To formalize the oracles, we need a new function symbol, $\mathtt{reveal}(\_) : \mathsf{message} \to \mathsf{message}$, and we add a few transitions to those in Example 4.1 as described below.

- Protocol Π_1 is defined such that the oracle always reveals the actual computed key of the requested session, if there is any: to a state $q^{k_1k_2k_3k_4}_{\ldots\ell_\alpha=1\ldots}$, we add the following transitions:

$$q^{k_1k_2k_3k_4}_{\ldots\ell_\alpha=1\ldots}, (\vec{N}), (\vec{x}) \xrightarrow{\theta_i} q^{k_1k_2k_3k_4}_{\ldots\ell_\alpha=2\ldots}, (\vec{N}, ()), \mathtt{m}(x_i)^{\mathsf{r}(n_i)}, (\vec{x}, x),$$

  where i runs through indices of $\vec{x} \equiv x_1, \ldots, x_m$ s.t. $N_i \neq ()$ and

$$\theta_i \equiv \mathsf{EQ}(\mathtt{reveal}(x), \mathtt{id}_\alpha)\ \&\ \mathsf{EQ}(\mathtt{to}(x_i), \mathtt{id}_\alpha).$$

  We order the transitions so that they are all applied after those in Example 4.1. The order of the transitions labeled with θ_i decreases with increasing i. If $q^{k_1k_2k_3k_4}_{\ldots\ell_\alpha=1\ldots} = q^{k_1k_2k_3k_4}_{\ldots\ell_\beta=1\ldots}$, then the transition corresponding to smaller of α and β has higher order. Moreover, to a state $q^{\ldots k_\alpha=2\ldots}_{\ell_1\ell_2\ell_3\ell_4}$, we add the following transitions:

$$q^{\ldots k_\alpha=2\ldots}_{\ell_1\ell_2\ell_3\ell_4}, (\vec{N}), (\vec{x}) \xrightarrow{\theta_{ih}} q^{\ldots k_\alpha=3\ldots}_{\ell_1\ell_2\ell_3\ell_4}, (\vec{N}, ()), \mathtt{m}(x_i)^{\mathsf{r}(n_h)}, (\vec{x}, x),$$

  where i and h run through all indices of $\vec{x} \equiv x_1, \ldots, x_m$ such that $N_h \neq ()$ with the restriction $h < i$, and

$$\begin{aligned}\theta_{ih} \equiv\ & \mathsf{EQ}(\mathtt{reveal}(x), \mathtt{id}_\alpha)\ \&\ \mathsf{EQ}(\mathtt{to}(x_i), \mathtt{id}_\alpha)\ \&\ \mathsf{EQ}(\mathtt{to}(x_h), \mathtt{id}_\alpha)\\ & \&\mathsf{not}(\mathsf{EQ}(\mathtt{act}(x_i), \mathtt{new}))\ \&\ \mathsf{EQ}(\mathtt{act}(x_h), \mathtt{new}).\end{aligned}$$

  We order the transitions so that they are all of higher order than those in Example 4.1, they decrease by α, and a transition labeled with θ_{ih} is higher for smaller i, and, within i, θ_{ih} is higher for smaller h. We also add

$$q^{k_1k_2k_3k_4}_{\ell_1\ell_2\ell_3\ell_4}, (\vec{N}), (\vec{x}) \xrightarrow{\theta_\alpha} \bar{q}, (\vec{N}, ()), \mathbf{0}, (\vec{x}, x),$$

  if $k_\alpha < 2$ and $\ell_\alpha < 1$ with $\theta_\alpha \equiv \mathsf{EQ}(\mathtt{reveal}(x), \mathtt{id}_\alpha)$, again with higher order than those transitions in Example 4.1.

In other words, in protocol Π_1, in each round, first it is checked if there was a $\mathtt{reveal}(x)$ request, and the oracle always reveals the key computed in session $\mathtt{reveal}(x)$ if such a key was computed. If there is no $\mathtt{reveal}(x)$ request, then Π_1 continues executing the DH protocol. The reason for the high number of transitions is that the oracle has to find the point where the key was computed.

- Protocol Π_2 is defined such that if the oracle request concerns a key that was computed in some session $\mathtt{id}_\alpha$, and there was another session $\mathtt{id}_\beta$ in which the same key was computed, then a g^c is revealed with a freshly generated random n. Otherwise, the computed key is revealed if there is any: to a state $q^{\dots k_\alpha=2\dots}_{\dots \ell_\beta=1\dots}$, besides the transitions of Π_1, we add the following transitions:

$$q^{\dots k_\alpha=2\dots}_{\dots \ell_\beta=1\dots}, (\vec{N}), (\vec{x}) \xrightarrow{\theta^i_{\gamma\delta\epsilon}} q^{\dots k_\alpha=3\dots}_{\dots \ell_\beta=2\dots}, (\vec{N}, (n)), \mathtt{g}(n_0)^{\mathtt{r}(n)}, (\vec{x}, x),$$

$$\begin{aligned}
\theta^1_{\gamma\delta\epsilon} \equiv\ & \mathtt{EQ}(\mathtt{reveal}(x), \mathtt{id}_\alpha)\ \&\ \mathtt{EQ}(\mathtt{to}(x_\gamma), \mathtt{id}_\alpha)\ \&\ \mathtt{EQ}(\mathtt{to}(x_\delta), \mathtt{id}_\beta)\\
&\&\mathtt{EQ}(\mathtt{to}(x_\epsilon), \mathtt{id}_\alpha)\ \&\ \mathtt{not}(\mathtt{EQ}(\mathtt{act}(x_\gamma), \mathtt{new}))\ \&\ \mathtt{EQ}(\mathtt{act}(x_\epsilon), \mathtt{new})\\
&\&\mathtt{EQ}(\mathtt{m}(x_\delta), \mathtt{g}(n_0)^{\mathtt{r}(n_\epsilon)})\ \&\ \mathtt{EQ}(\mathtt{m}(x_\gamma), \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)}),\\
\theta^2_{\gamma\delta\epsilon} \equiv\ & \mathtt{EQ}(\mathtt{reveal}(x), \mathtt{id}_\beta)\ \&\ \mathtt{EQ}(\mathtt{to}(x_\gamma), \mathtt{id}_\alpha)\ \&\ \mathtt{EQ}(\mathtt{to}(x_\delta), \mathtt{id}_\beta)\\
&\&\mathtt{EQ}(\mathtt{to}(x_\epsilon), \mathtt{id}_\alpha)\ \&\ \mathtt{not}(\mathtt{EQ}(\mathtt{act}(x_\gamma), \mathtt{new}))\ \&\ \mathtt{EQ}(\mathtt{act}(x_\epsilon), \mathtt{new})\\
&\&\mathtt{EQ}(\mathtt{m}(x_\delta), \mathtt{g}(n_0)^{\mathtt{r}(n_\epsilon)})\ \&\ \mathtt{EQ}(\mathtt{m}(x_\gamma), \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)}),
\end{aligned}$$

  and $\gamma > \delta > \epsilon$.The new transitions are ordered so that they have a higher order than the transitions in Π_1. Amongst the new transitions of Π_2, the transitions are ordered by decreasing α, then decreasing β, then decreasing i, then decreasing γ, then decreasing δ and decreasing ϵ. These checks ensure that if there is a session where they computed the matching keys, then a newly generated random key $\mathtt{g}(n_0)^{\mathtt{r}(n)}$ is revealed.

Note, the oracle requests do not interfere with the protocol. Their sole purpose is to model secrecy of some of the computed keys, namely, those for which there is a session with a matching key. The next theorem states that such keys satisfy real-or-random secrecy.

Proposition 8.1. *The above two protocols, Π_1 and Π_2, allowing two parallel sessions for the DH key exchange protocol, are computationally indistinguishable as long as the group scheme satisfies the DDH assumption.*

Proof. Consider first Π_1, and let us make the following observation. When the protocol is folded, there are if then else branchings for each conjunct in the conditions θ, including for those that appear in the oracle requests. However, in θ_i above in the oracle move, only $\mathtt{EQ}(\mathtt{reveal}(x), \mathtt{id}_\alpha)$ is a new condition, the condition $\mathtt{EQ}(\mathtt{to}(x_i), \mathtt{id}_\alpha)$ already appeared earlier in the execution, so there is already a branching according to the latter. By `IfMorph` and `IfIdemp`, just as in Example 7.9, such additional branching can be removed while the output $\mathtt{m}(x_i)^{\mathtt{r}(n_i)}$, takes the value of the form $\mathtt{m}(f_i(\phi_i))^{\mathtt{r}(n_i)}$ where $\mathtt{EQ}(\mathtt{to}(f_i(\phi_i)), \mathtt{id}_\alpha)$ is satisfied. Only the branching according to $\mathtt{EQ}(\mathtt{reveal}(x), \mathtt{id}_\alpha)$ remains in the oracle step. When $\mathtt{id}_\alpha$ is a Responder session, $\mathtt{m}(f_i(\phi_i))^{\mathtt{r}(n_i)}$ is the key computed in this session and revealed by the oracle. The same is true for θ_{ih}, but there $\mathtt{id}_\alpha$ is an Initiator session and the oracle output is accordingly $\mathtt{m}(f_i(\phi_i))^{\mathtt{r}(n_h)}$, where $\mathtt{r}(n_h)$ is computed initially in this session, and $f_i(\phi_i)$ is the message that is supposed to be coming from the responder. In Π_2, conditions $\mathtt{EQ}(\mathtt{m}(x_\delta), \mathtt{g}(n_0)^{\mathtt{r}(n_\epsilon)})$ and $\mathtt{EQ}(\mathtt{m}(x_\gamma), \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)})$ are also new in the oracle step, so these branchings cannot be removed. As $\mathtt{m}(x_\gamma)^{\mathtt{r}(n_\epsilon)}$ is the key computed in the Initiator session $\mathtt{id}_\alpha$, while $\mathtt{m}(x_\delta)^{\mathtt{r}(n_\delta)}$ is the key computed in Responder session $\mathtt{id}_\beta$, conditions $\mathtt{EQ}(\mathtt{m}(x_\delta), \mathtt{g}(n_0)^{\mathtt{r}(n_\epsilon)})$ and $\mathtt{EQ}(\mathtt{m}(x_\gamma), \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)})$ make sure that both are $\mathtt{g}(n_0)^{\mathtt{r}(n_\delta)\mathtt{r}(n_\epsilon)}$.

With this understanding in mind, consider a protocol Π''_2, which is like Π_2, but in the transition, we replace the output $\mathtt{g}(n_0)^{\mathtt{r}(n)}$ by $\mathtt{m}(x_\delta)^{\mathtt{r}(n_\delta)}$ for $i = 2$, and by $\mathtt{m}(x_\gamma)^{\mathtt{r}(n_\epsilon)}$ for $i = 1$. This means that Π''_2 outputs the same exact messages as Π_1, ignoring the additional branching. Considering the frames, that means that $\Phi(\mathrm{fold}(\Pi_1)) \sim \Phi(\mathrm{fold}(\Pi''_2))$ by several applications of axiom `IfSame`. Then, consider the protocol Π'_2, which we obtain from Π_2 by replacing the output $\mathtt{g}(n_0)^{\mathtt{r}(n)}$ by

$\mathsf{g}(n_0)^{\mathsf{r}(n_\delta)\mathsf{r}(n_\epsilon)}$. Then, according to the previous paragraph, using the results of Examples 7.3 and 7.4, we have $\Phi(\text{fold}(\Pi_2'')) \sim \Phi(\text{fold}(\Pi_2'))$.

The only thing left to prove is $\Phi(\text{fold}(\Pi_2')) \sim \Phi(\text{fold}(\Pi_2))$. This relies mainly on the DDH axiom and the IfBranch axiom: The only difference between $\Phi(\text{fold}(\Pi_2'))$ and $\Phi(\text{fold}(\Pi_2))$ is that some of the final sent messages are $\mathsf{g}(n_0)^{\mathsf{r}(n_\delta)\mathsf{r}(n_\epsilon)}$ in the first, while $\mathsf{g}(n_0)^{\mathsf{r}(n)}$ in the second. We cannot immediately use the DDH axiom, because the values of δ and ϵ may vary from branch to branch. Considering just a single branch of Π_2', the complete list of messages that have been sent looks like $\mathsf{G}(\mathsf{r}(n_0)), \mathsf{g}(\mathsf{r}(n_0)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, \ldots, g^{\mathsf{r}(n_4)}, \mathsf{g}(n_0)^{\mathsf{r}(n_\delta)\mathsf{r}(n_\epsilon)}$. Because of the DDH assumption, FreshInd, FuncApp, we have

$$\begin{aligned}&\mathsf{G}(\mathsf{r}(n_0)), \mathsf{g}(\mathsf{r}(n_0)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, \ldots, \mathsf{g}(n_0)^{\mathsf{r}(n_4)}, \mathsf{g}(n_0)^{\mathsf{r}(n_\delta)\mathsf{r}(n_\epsilon)}\\ &\sim \mathsf{G}(\mathsf{r}(n_0)), \mathsf{g}(\mathsf{r}(n_0)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, \ldots, \mathsf{g}(n_0)^{\mathsf{r}(n_4)}, \mathsf{g}(n_0)^{\mathsf{r}(n)}.\end{aligned}$$

All tests θ in the protocol definition are applied only on messages sent by the adversary (which are functions applied on public terms) and $\mathsf{g}(n_0)^{\mathsf{r}(n_1)}, \ldots, g^{\mathsf{r}(n_4)}$. Hence, for such a test θ, by FuncApp and Restr, we also have

$$\begin{aligned}&\theta, \mathsf{G}(\mathsf{r}(n_0)), \mathsf{g}(\mathsf{r}(n_0)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, \ldots, \mathsf{g}(n_0)^{\mathsf{r}(n_4)}, \mathsf{g}(n_0)^{\mathsf{r}(n_\delta)\mathsf{r}(n_\epsilon)}\\ &\sim \theta, \mathsf{G}(\mathsf{r}(n_0)), \mathsf{g}(\mathsf{r}(n_0)), \mathsf{g}(n_0)^{\mathsf{r}(n_1)}, \ldots, \mathsf{g}(n_0)^{\mathsf{r}(n_4)}, \mathsf{g}(n_0)^{\mathsf{r}(n)}.\end{aligned}$$

We can add all the tests along the branch, and we can do the same for all branches, with different δ and ϵ. Using axiom IfBranch numerous times, all the equivalent branches can be folded into branching terms, giving us $\Phi(\text{fold}(\Pi_2')) \sim \Phi(\text{fold}(\Pi_2))$. This completes the proof. □

Remark 4. Of course, an automated proof would work directly transforming the frames, not through transforming the protocols. Extension to proofs for higher (but bounded) number of sessions is a straightforward matter; only the formulas would be longer. The proof for the key exchange with more than two parties is also entirely analogous once the DDH property for more parties is derived. We did this for three parties in Example 7.10, and for more parties the derivation is similar.

9 DIGITAL SIGNATURES

To continue to demonstrate the usability of our technique, we also consider authentication that signatures can deliver. In this section, we introduce an axiom that formalizes UF-CMA secure digital signatures (see Section 12.2 of Reference [24]). In the next section, we demonstrate how to use it together with the core axioms to verify an authenticated DH key exchange. Accordingly, we shall also include in $\mathcal{F}$ the following function symbols:

$$\begin{array}{lrcl}
\mathsf{k}(_): & \mathsf{message} & \to & \mathsf{message},\\
\mathsf{r_s}(_): & \mathsf{message} & \to & \mathsf{message},\\
\mathsf{sign}(_,_,_): & \mathsf{message} \times \mathsf{message} \times \mathsf{message} & \to & \mathsf{message},\\
\mathsf{ver}(_,_,_): & \mathsf{message} \times \mathsf{message} \times \mathsf{message} & \to & \mathsf{bool}.
\end{array}$$

Here, k(_) denotes the public-key secret-key pair generation algorithm. An honest key looks like $\mathsf{k}(n)$, where n is a name and $\mathsf{pk}(x) \stackrel{\text{def}}{\equiv} \pi_1(\mathsf{k}(x))$ and $\mathsf{sk}(x) \stackrel{\text{def}}{\equiv} \pi_2(\mathsf{k}(x))$ are the public verification key and secret signing key parts of $\mathsf{k}(x)$, respectively. To allow for randomized signatures, we introduce a symbol $\mathsf{r_s}(\_)$ for random seed generation. $\mathsf{sign}(y, z, r)$ is the message z signed with secret key y and a random seed r. $\mathsf{ver}(y, z, u)$ is the verification of signature u on the message z with the public key y. The co-domain of the function symbol ver is bool as the computational interpretation of ver outputs a value in $\{0, 1\}$.

The signature scheme must satisfy two conditions:

- *Correctness:* If a message signed with $\mathsf{sk}(x)$ is verified with the corresponding $\mathsf{pk}(x)$, then the verification algorithm outputs 1. This is captured by the axiom schema:

$$\mathsf{ver}(\mathsf{pk}(x), t, \mathsf{sign}(\mathsf{sk}(x), t, \mathsf{r_s}(y))) = \textbf{\textit{true}}.$$

- *Existential unforgeability under adaptively chosen message attacks (UF-CMA secure):* Informally, this is the security requirement for digital signatures and says that a PPT attacker should not be able to forge a signature on any message chosen by the attacker, even after requesting an oracle to show the signatures of at most polynomial number of messages adaptively chosen by him. The interested reader can find the precise definition in Section 12.2 of Reference [24].

  We now state an axiom schema that captures UF-CMA security. Let n be a name and let t, u be closed terms such that all occurrences of $\mathsf{sk}(n)$ in t, u can be enumerated as $\mathsf{sign}(\mathsf{sk}(n), t_1, \mathsf{r_s}(n_1)), \mathsf{sign}(\mathsf{sk}(n), t_2, \mathsf{r_s}(n_2)), \ldots, \mathsf{sign}(\mathsf{sk}(n), t_\ell, \mathsf{r_s}(n_\ell))$. The term $\mathsf{sk}(n)$ does not occur in any other form in t, u, and all other occurrences of n in t, u are of the form $\mathsf{pk}(n)$. Let $b^0_{t,u}, b^1_{t,u}, \ldots b^\ell_{t,u}$ be defined recursively as

$$\begin{aligned} b^0_{t,u} &\stackrel{\text{def}}{\equiv} \textbf{\textit{false}}, \\ b^j_{t,u} &\stackrel{\text{def}}{\equiv} \text{if } \mathsf{EQ}(t, t_j) \text{ then } \mathsf{ver}(\mathsf{pk}(n), t, u) \text{ else } b^{j-1}_{t,u}\ . \end{aligned}$$

  Then, the axiom schema is $\mathsf{ver}(\mathsf{pk}(n), t, u) = b^\ell_{t,u}$. That is, if t is one of t_j, the signature of which appears in t or u, then the right-hand side outputs $\mathsf{ver}(\mathsf{pk}(n), t, u)$. If t is neither of t_j, then the right-hand side outputs ***false***, expressing the idea that no signature of a new t can be created. We shall henceforth refer to this axiom schema UF-CMA.

Proposition 9.1. *If the interpretation of* $(\mathsf{k}, \mathsf{ver}, \mathsf{sign}, \pi_1, \pi_2)$ *satisfies the UF-CMA property, then the* UF-CMA *axiom is sound. Conversely, if there is a constant* $\ell \in \mathbb{N}$ *and an UF-CMA attack* $\mathcal{A}$ *against the interpretation of* $(\mathsf{k}, \mathsf{ver}, \mathsf{sign})$ *such that the number of oracle queries* $\mathcal{A}$ *makes does not exceed* ℓ *for any* η*, then* UF-CMA *axiom is violated in some computational model (with the given interpretation of* $(\mathsf{k}, \mathsf{ver}, \mathsf{sign}, \pi_1, \pi_2)$*).*

Proof. We proceed by contradiction. Assume that there are closed terms t, u and a computational model $\mathcal{M}^c$ such that $\mathcal{M}^c \not\models \mathsf{EQ}(\mathsf{ver}(\mathsf{pk}(n), t, u), b^\ell_{t,u}) \sim \textbf{\textit{true}}$ where $b^\ell_{t,u}$ is defined as in the axiom UF-CMA. This means that there is a Turing machine $\mathcal{A}$ that runs in polynomial time in the security parameter η such that

$$\begin{aligned} Adv^{\mathcal{A}}(\eta) = |&\mathbf{Prob}\{\rho :\ \mathcal{A}([\![\textbf{\textit{true}}]\!]_{\rho,\eta}; \rho_2) = 1\} - \\ &\mathbf{Prob}\{\rho :\ \mathcal{A}([\![\mathsf{EQ}(\mathsf{ver}(\mathsf{pk}(n), t, u), b^\ell_{t,u})]\!]_{\rho,\eta}; \rho_2) = 1\}| \end{aligned}$$

is a non-negligible function in η.

By definition, $[\![\textbf{\textit{true}}]\!]_{\rho,\eta} = 1$ and $[\![\mathsf{EQ}(\mathsf{ver}(\mathsf{pk}(n), t, u), b^\ell_{t,u})]\!]_{\rho,\eta} = 1$ whenever $[\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} = [\![b^\ell_{t,u}]\!]_{\rho,\eta}$. Thus, $Adv^{\mathcal{A}}(\eta) \leq \mathbf{Prob}\{\rho :\ [\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} \neq [\![b^\ell_{t,u}]\!]_{\rho,\eta}\}$.

Thanks to the semantics of if _ then _ else _ , we have that the set $\{\rho :\ [\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} \neq [\![b^\ell_{t,u}]\!]_{\rho,\eta}\}$ is exactly the set

$$F(\eta) = \left\{ \rho :\ [\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} = 1, \bigwedge_{i=1}^{\ell} [\![t]\!]_{\rho,\eta} \neq [\![t_j]\!]_{\rho,\eta} \right\}.$$

Since $Adv^{\mathcal{A}}(\eta)$ is a non-negligible function in η, $\mathbf{Prob}\{F(\eta)\}$ is a non-negligible function in η.

Then an adversary $\mathcal{B}$ can win the UF-CMA game against $\mathsf{pk}(n)$ as follows. On the security parameter η, $\mathcal{B}$ is given $[\![\mathsf{pk}(n)]\!]_{\rho,\eta}$ by the oracle. $\mathcal{B}$ generates an interpretation of names that occur in t, u according to $\mathcal{M}^c$. Then $\mathcal{B}$ computes $[\![t]\!]_{\rho',\eta}, [\![u]\!]_{\rho',\eta}$ using its interpretation for the names; whenever it needs to compute a signature $\mathsf{sign}(\mathsf{sk}(n), t_i, \mathsf{r_s}(n_i))$, it consults the oracle. It is easy to see that the probability $\mathcal{B}$ wins the UF-CMA game is exactly $\mathbf{Prob}\{F(\eta)\}$, which is a non-negligible function in η.

Proving the converse is equally easy. Let us consider an UF-CMA attacker $\mathcal{A}$ on the given interpretation of $\mathsf{k}, \mathsf{sign}, \mathsf{ver}, \pi_1, \pi_2$ that succeeds with non-negligible probability and makes at most ℓ oracle queries. Let $[\![\mathsf{k}]\!], [\![\mathsf{sign}]\!], [\![\mathsf{ver}]\!], [\![\pi_1]\!], [\![\pi_2]\!]$ be the interpretation of $\mathsf{k}, \mathsf{sign}, \mathsf{ver}, \pi_1, \pi_2$.

Fix a name n and function symbols $f_0, f_1, \ldots, f_{\ell+1}, f'_{\ell+1} \in \mathcal{G}$. Let $t_1, t_2, \ldots, t_{\ell+1}, t, u$ be defined as follows:

$$\begin{aligned}
t_1 &\stackrel{\text{def}}{\equiv} f_0(\mathsf{pk}(n))\\
t_{i+1} &\stackrel{\text{def}}{\equiv} f_{i+1}(\mathsf{pk}(n), \mathsf{sign}(\mathsf{sk}(n), t_i, \mathsf{r_s}(n_i)), \ldots, \mathsf{sign}(\mathsf{sk}(n), t_1, \mathsf{r_s}(n_1)))\\
t &\stackrel{\text{def}}{\equiv} t_{\ell+1}\\
u &\stackrel{\text{def}}{\equiv} f'_{\ell+1}(\mathsf{pk}(n), \mathsf{sign}(\mathsf{sk}(n), t_\ell, \mathsf{r_s}(n_\ell)), \ldots, \mathsf{sign}(\mathsf{sk}(n), t_1, \mathsf{r_s}(n_1))).
\end{aligned}$$

Fix the interpretation of $f_0, f_1, \ldots, f_{\ell+1}, f'_{\ell+1}$ as follows. $[\![f_0]\!]$ is the Turing Machine that on input m and tapes $\rho_1; \rho_2$ simulates the attacker $\mathcal{A}$ until it prepares the message query to be submitted to the signing oracle. At that point $[\![f_0]\!]$ outputs the actual query and stops. For $0 < j \leq \ell$, $[\![f_j]\!]$ is the Turing Machine that on input m, s_1, s_2, s_{j-1} simulates the attacker $\mathcal{A}$ until it prepares the j-th query to be submitted to the signing oracle with one minor modification: whenever $\mathcal{A}$ submits the ith query to the oracle for $i < j$ and gets the signature on the query; $[\![f_j]\!]$ does not query the oracle and uses s_i instead of the signature. $[\![f_j]\!]$ outputs the jth query to be submitted to the oracle and stops. In a similar fashion, $[\![f_{\ell+1}]\!]$ simulates $\mathcal{A}$ until $\mathcal{A}$ is ready to output a message and a *claimed* signature on the message. $[\![f_{\ell+1}]\!]$ outputs *just* the message and stops. Likewise, $[\![f'_{\ell+1}]\!]$ simulates $\mathcal{A}$ until $\mathcal{A}$ is ready to output a message and a claimed signature on the message, outputs *just* the signature part and stops. It is easy to see that the computational model $\mathcal{M}^c$ with this interpretation of $\mathsf{k}, \mathsf{sign}, \mathsf{ver}, \pi_1, \pi_2, f_0, f_1, \ldots f_{\ell+1}, f'_{\ell+1}$ violates the axiom schema: Let $F(\eta)$ be again defined as

$$F(\eta) = \left\{ \rho : \ [\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} = 1, \bigwedge_{i=1}^{\ell} [\![t]\!]_{\rho,\eta} \neq [\![t_j]\!]_{\rho,\eta} \right\}.$$

Since $\mathcal{A}$ is assumed to break the UF-CMA security, the set $F(\eta)$ is non-negligible. But, it is again easy to see that

$$F(\eta) = \{\rho : \ [\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} \neq [\![b^{\ell}_{t,u}]\!]_{\rho,\eta}\}.$$

Hence, if we define $\mathcal{B}$ to be the algorithm that outputs its input, then we have that

$$\begin{aligned}
Adv^{\mathcal{B}}(\eta) = &|\mathbf{Prob}\{\rho : \ \mathcal{B}([\![\boldsymbol{true}]\!]_{\rho,\eta}; \rho_2) = 1\} -\\
&\mathbf{Prob}\{\rho : \ \mathcal{B}([\![\mathsf{EQ}(\mathsf{ver}(\mathsf{pk}(n), t, u), b^{\ell}_{t,u})]\!]_{\rho,\eta}; \rho_2) = 1\}|
\end{aligned}$$

is non-negligible. Hence, the converse follows. □

The converse of this proposition means that our axiom is as tight as possible as the technique works only for bounded number of sessions and hence bounded number of signatures.

10 AUTHENTICATED DIFFIE-HELLMAN KEY EXCHANGE

We apply our core axioms to rather different purpose: *authentication*. We consider an authenticated Diffie-Hellman key exchange protocol, which is a simplified version of the station-to-station protocol. Note that the original station-to-station protocol contains key-confirmation as well using encryption; we omit that now to keep syntax simple. Our version of the protocol is the following:

- A group description G and a group generator element g are generated honestly, according to a randomized algorithm, and made public. Public verification key, secret signing key pairs are generated honestly for honest agents and the verifification keys are made public.
- The Initiator, A, selects a responder B, generates a random a in $\mathbb{Z}_{|g|}$ and sends $\langle A, B, g^a\rangle$.
- The Responder, B, receives g^a, generates a random b in $\mathbb{Z}_{|g|}$ and sends $\langle g^b, \mathsf{sign}(sk_B, \langle A, g^b, g^a\rangle, r)\rangle$, and computes $(g^a)^b$.
- The Initiator receives $\langle g^b, \mathsf{sign}(sk_B, \langle A, g^b, g^a\rangle, r)\rangle$, verifies the signature, computes $(g^b)^a$, and sends $\mathsf{sign}(sk_A, \langle B, g^a, g^b\rangle, r')$.
- The Responder receives $\mathsf{sign}(sk_A, \langle B, g^a, g^b\rangle, r')$, verifies the signature, and outputs acc.

Here, we can think of triples being constructed from pairs: $\langle x, y, z\rangle := \langle x, \langle y, z\rangle\rangle$, and for projecting on the components, let $\tau_1 := \pi_1$, $\tau_2 := \pi_1 \circ \pi_2$, $\tau_3 := \pi_2 \circ \pi_2$.

Real-or-random secrecy for the shared keys can be verified the same way as for the DH protocol, no new axioms are needed.

We shall show that with the help of the UF-CMA axiom, we can also prove authentication of the authenticated key exchange. We concentrate on the responder's non-injective authentication of the initiator, and the initiator's authentication of the responder can be handled similarly. We assume that there are two honest agents A and B. Public key, secret key pairs $(\mathsf{pk}(n_A), \mathsf{sk}(n_A))$ and $(\mathsf{pk}(n_B), \mathsf{sk}(n_B))$ are generated honestly for A and B, respectively. For simplicity, we assume that all sessions of A are initiator sessions and all sessions of B are responder sessions. We also assume that there are no other agents (this is actually not really a restriction, because the adversary can simulate other agents), but there can be other agent id's and associated public keys. We assume that finite number of agent id—public key pairs $(A, \mathsf{pk}(n_A))$, $(B, \mathsf{pk}(n_B))$ (for honest agents) and $(Q_1, \mathsf{pk}(c_1))$, $(Q_2, \mathsf{pk}(c_2))$, ... (for dishonest agents) are publicly available to associate agent id's with public keys. The function symbols c_1, c_2, etc are adversarial constants. We assume that the agent ids $A, B, Q_1, Q_2, \ldots$ are pairwise distinct.

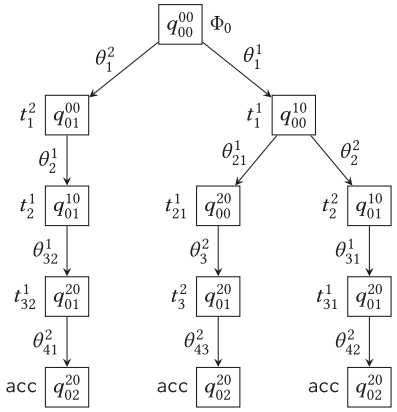
When A is instructed to start an initiator session then it is given the agent id of the responder. A checks that the agent name is in the list of available agent ids and can then extract the corresponding public keys. To simplify presentation, we assume the following abbreviations to accomplish the above:

$$check(Q) = \mathsf{EQ}(Q, A) \parallel \mathsf{EQ}(Q, B) \parallel \mathsf{EQ}(Q, Q_1) \parallel \cdots$$

and

$$\begin{aligned} pkey(Q) \quad = \quad & \text{if } \mathsf{EQ}(Q, A) \text{then } \mathsf{pk}(n_A) \\ & \text{else if } \mathsf{EQ}(Q, B) \text{ then } \mathsf{pk}(n_B) \\ & \qquad \text{else if } \mathsf{EQ}(Q, Q_1) \text{ then } \mathsf{pk}(c_1) \\ & \qquad\qquad \vdots \\ & \qquad\qquad\qquad \text{else } \mathbf{0}. \end{aligned}$$

The responder similarly checks if the initiator in the first message is in the list of available agent ids.



$\Phi_0 \equiv \mathsf{G}(n), \mathsf{g}(n), A, \mathsf{pk}(n_A), B, \mathsf{pk}(n_B), Q_1, \mathsf{pk}(c_1), \cdots$
$\theta_i^1 \equiv (\mathsf{EQ}(\mathsf{to}(x_i), \mathtt{id}_1)\ \&\ \mathsf{EQ}(\mathsf{act}(x_i), \mathsf{new})\ \&\ check(\mathsf{m}(x_i))$
$g^a \equiv \mathsf{g}(n)^{\mathsf{r}(n_1)};\ g^b \equiv \mathsf{g}(n)^{\mathsf{r}(n_2)}$
$t_i^1 \equiv \langle A, \mathsf{m}(x_i), g^a \rangle$
$\theta_{ij}^1 \equiv (\mathsf{EQ}(\mathsf{to}(x_i), \mathtt{id}_1)\&$
$\mathsf{ver}(pkey(\mathsf{m}(x_j)), \langle A, \pi_1(\mathsf{m}(x_i)), g^a \rangle, \pi_2(\mathsf{m}(x_i))$
$t_{ij}^1 \equiv \mathsf{sign}(\mathsf{sk}(n_A), \langle \mathsf{m}(x_j), g^a, \pi_1(\mathsf{m}(x_i)) \rangle, \mathsf{r_s}(n_3))$
$\theta_i^2 \equiv (\mathsf{EQ}(\mathsf{to}(x_i), \mathtt{id}_2)\ \&\ check(\tau_1(\mathsf{m}(x_i)))\ \&\ \mathsf{EQ}(\tau_2(\mathsf{m}(x_i)), B)$
$t_i^2 \equiv \langle g^b, \mathsf{sign}(\mathsf{sk}(n_B), \langle \tau_1(\mathsf{m}(x_i)), g^b, \tau_3(\mathsf{m}(x_i)) \rangle, \mathsf{r_s}(n_4)) \rangle$
$\theta_{ij}^2 \equiv (\mathsf{EQ}(\mathsf{to}(x_i), \mathtt{id}_2)\ \&$
$\mathsf{ver}(pkey(\tau_1(m(x_j))), \langle B, \tau_3(m(x_j)), g^b \rangle, \mathsf{m}(x_i))$

Fig. 2. Authenticated Diffie-Hellman key exchange protocol with two agents A and B.

We present a transition diagram that represents the authenticated DH protocol with two participants, an initiator A and a responder B, in Figure 2. For simplicity, we illustrate the case when there is only one initiator session $\mathtt{id}_1$ and one responder session $\mathtt{id}_2$. The diagram illustrates three possible branches of the protocol. The right-most branch simulates the situation where the initiator A moves first and then the responder B moves next, whereas the left-most branch simulates the other scenario that B moves first followed by A. The middle branch illustrates the scenario when all the moves of A happen before the moves of B. The frame Φ_0 represents the initial knowledge of the attacker, which includes a description of a cyclic group $\mathsf{G}(n)$, a generator of the group $\mathsf{g}(n)$, the honest agent-ids, A, B, and their respective public keys $\mathsf{pk}(n_A)$ and $\mathsf{pk}(n_B)$. The initial frame also includes finite number of ids of dishonest agents, $Q_1, Q_2, \ldots$ and their public keys $\mathsf{pk}(c_1), \mathsf{pk}(c_2), \ldots$, respectively. On the input message x_i, the conditions θ_i^1 checks if the input message x_i is for A and he has instructed to start a new session with agent $\mathsf{m}(x_i)$. This transition also checks that if the received agent id $\mathsf{m}(x_i)$ is valid. If the checks succeeded, then A extracts the public key $pkey(\mathsf{m}(x_i))$ of the agent $\mathsf{m}(x_i)$. t_i^1 represents the A's initial message, i.e., the triple consisting of the initiator agent-id A, the received agent-id $\mathsf{m}(x_i)$, and the computed group element g^a in the protocol. θ_{ij}^1 checks that the received message x_i is for A and verifies that it is a valid response by checking the signature using the public key $pkey(\mathsf{m}(x_j))$, which he has computed in the previous round of the protocol. t_{ij}^1 represents the final message for A in the protocol, i.e., the signed message $\mathsf{sign}(\mathsf{sk}(n_A), \langle \mathsf{m}(x_j), g^a, \pi_1(\mathsf{m}(x_i)) \rangle, \mathsf{r_s}(n_3))$ of A using his own secret key $\mathsf{sk}(n_A)$ and a random seed $\mathsf{r_s}(n_3)$. Similarly, in θ_i^2, we check that the message x_i is for B, $\tau_1(\mathsf{m}(x_i))$ is a valid agent, and that B is the intended responder. If these checks succeeded, then the responder B extracts the public key $pkey(\tau_1(\mathsf{m}(x_i)))$ of the agent $\tau_1(\mathsf{m}(x_i))$. t_i^2 is the initial response by B, i.e., the pair that consists of the group element g^b and the signed message $\mathsf{sign}(\mathsf{sk}(n_B), \langle \tau_1(\mathsf{m}(x_i)), b, \tau_3(\mathsf{m}(x_i)) \rangle, \mathsf{r_s}(n_4))$. θ_{ij}^2 checks that if the message x_i is for B and checks that the received message is the expected final message of the protocol by verifying the signature on the message using public key of the initiator $pkey(\tau_1(\mathsf{m}(x_j)))$. Of course, at the end of the protocol, the responder B also sends out the message acc to indicate that the protocol has been completed. We skip other branches for lack of space.

Now, we explain how authentication can be modeled in our framework. Please note that we are considering two initiator sessions for A and two responder sessions for B in the following. Our methods can be easily extended to any fixed number of sessions. Responder's authentication of the initiator means that if B received and verified a message that looked like $\mathsf{sign}(\mathsf{sk}(n_A), \langle B, y, g^b \rangle, r')$

for some input y, then A has a matching initiator session: A has a session in which he sent $\mathsf{sign}(\mathsf{sk}(n_A), \langle Q, g^a, x\rangle, r')$ for some x message and agent Q (and as implied by A's role also sent $\langle A, Q, g^a\rangle$ before that) before B received it on the same branch, which is the same branch that A received a message that looked like $\langle x, z\rangle$, the verification $\mathsf{ver}(pkey(Q), \langle A, x, g^a\rangle, z)$ succeeded, and $\mathsf{EQ}(x, g^b)$ & $\mathsf{EQ}(y, g^a)$ & $\mathsf{EQ}(Q, B)$ is satisfied on this branch.

There are various possibilities to express responder's authentication of the initiator in our language, we present one. Namely, similarly to our modeling of secrecy, we can define an oracle query that takes a session $\mathtt{id}_\alpha$ as input, and if $\mathtt{id}_\alpha$ is a completed responder session and there is no matching initiator session in the above sense then the oracle outputs 1 (that is, ***true*** symbolically), meaning there is an attack against authentication. Otherwise, it outputs 0 (that is, ***false*** symbolically). Let the protocol that ends with such an oracle query be called Π_1^{auth}. We can also define Π_2^{auth} such that the oracle always outputs 0. These oracles can be formalized as in the case of secrecy. Then, the authentication property can be formalized as

$$\Phi\Big(\mathsf{fold}\Big(\Pi_1^{\mathsf{auth}}\Big)\Big) = \Phi\Big(\mathsf{fold}\Big(\Pi_2^{\mathsf{auth}}\Big)\Big).$$

Observe that we used equality and not indistinguishability. This means that Π_1^{auth} cannot output 1 with non-negligible probability.

Proposition 10.1. *Let* Π_1^{auth} *and* Π_2^{auth} *be the two protocols as defined above. Assuming the signature scheme satisfies the* UF-CMA *assumption,* $\Phi(\mathsf{fold}(\Pi_1^{\mathsf{auth}})) = \Phi(\mathsf{fold}(\Pi_2^{\mathsf{auth}}))$.

Proof (Sketch). If on a branch of $\Phi(\mathsf{fold}(\Pi_1^{\mathsf{auth}}))$ there is a ***true*** as the final output, then by the definition of the oracle this branch lies on the true side of the branching where the condition $\mathsf{EQ}(\tau_1(\mathtt{m}(f_i(\phi_i))), A)$ and the condition $\mathsf{ver}(pkey(\tau_1(\mathtt{m}(f_i(\phi_i)))), \langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle, \mathtt{m}(f_j(\phi_j)))$ at the last move of the responder are true. Here, $f_i(\phi_i)$ is the message that B is supposed to have received from A earlier and whose body is supposed to be $\langle A, B, g^a\rangle$, while the message $f_j(\phi_j)$ is supposed to be from A with body $\mathsf{sign}(\mathsf{sk}(n_A), \langle B, g^a, g^b\rangle, r')$. Since $\mathsf{EQ}(\tau_1(\mathtt{m}(f_i(\phi_i))), A)$ is true on this branch, we can always replace $\tau_1(\mathtt{m}(f_i(\phi_i)))$ by A and $pkey(\tau_1(\mathtt{m}(f_i(\phi_i))))$ by $\mathsf{pk}(n_A)$ on this branch. According to UF-CMA and EqCong, $\mathsf{ver}(\mathsf{pk}(n_A), \langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle, \mathtt{m}(f_j(\phi_j)))$ can be rewritten as a branching term, which gives ***false*** (and hence the final output is also ***false*** by axioms IfMorph and IfIdemp and the definition of the oracle) unless ϕ_j (and hence the earlier ϕ_i) contains a term $\mathsf{sign}(\mathsf{sk}(n_A), t, r')$ for some t such that $\mathsf{EQ}(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle, t)$ evaluates to true. By the role of A, $\mathsf{sign}(\mathsf{sk}(n_A), t, r')$ must be of the form $\mathsf{sign}(\mathsf{sk}(n_A), \langle \mathtt{m}(f_\ell(\phi_\ell)), g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle, r')$ for an a that A generated at the beginning of the role in response to the message $f_\ell(\phi_\ell)$. Here, $f_h(\phi_h)$ is supposed to be the initial response received by A. In the subsequent proof, we abbreviate $\mathtt{m}(f_\ell(\phi_\ell))$ by Q. Now, we are on the true side of the $\mathsf{EQ}(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle, \langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)$ branching. In the final oracle step, there is a branching according to $\mathsf{EQ}(\tau_3(\mathtt{m}(f_i(\phi_i))), g^a)$ & $\mathsf{EQ}(\tau_1(\mathtt{m}(f_h(\phi_h))), g^b)$ & $\mathsf{EQ}(Q, B)$, which must fail for the oracle to output ***true***, because otherwise there is a matching session. In the rest of the argument, we show that if we are on the true side of $\mathsf{EQ}(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle, \langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)$, then in the final oracle step the branching $\mathsf{EQ}(\tau_3(\mathtt{m}(f_i(\phi_i))), g^a)$ & $\mathsf{EQ}(\tau_1(\mathtt{m}(f_h(\phi_h))), g^b)$ & $\mathsf{EQ}(Q, B)$ can be replaced with ***true***, and hence the final output is always ***false***. By the equational theory of pairs, and congruence of equality, the term $\mathsf{EQ}(\tau_3(\mathtt{m}(f_i(\phi_i))), g^a)$ &$\mathsf{EQ}(\tau_1(\mathtt{m}(f_h(\phi_h))), g^b)$ & $\mathsf{EQ}(Q, B)$ can be rewritten as

$$\begin{aligned}&\mathsf{EQ}(\tau_1(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle), \tau_1(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle)) \;\&\\ &\mathsf{EQ}(\tau_2(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle), \tau_2(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)) \;\&\\ &\mathsf{EQ}(\tau_3(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle), \tau_3(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle)).\end{aligned}$$

As we are on the true side of $\mathsf{EQ}(\langle B, \tau_3(\mathtt{m}(f_i(\phi_i))), g^b\rangle, \langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)$, using Example 7.3 and Example 7.4 the above checks can be replaced by

$$\begin{array}{l}\mathsf{EQ}(\tau_1(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle), \tau_1(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)) \;\&\\ \mathsf{EQ}(\tau_2(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle), \tau_2(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)) \;\&\\ \mathsf{EQ}(\tau_3(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle), \tau_3(\langle Q, g^a, \tau_1(\mathtt{m}(f_h(\phi_h)))\rangle)),\end{array}$$

which in turn can be replaced by ***true*** by EqRefl and EqCong. This means $\Phi(\mathsf{fold}(\Pi_1^{\mathsf{auth}}))$ is equal to a frame where all final outputs are ***false***. Then, by axiom IfSame and congruence, all branchings of the final oracle step can be collapsed, and thus we obtain $\Phi(\mathsf{fold}(\Pi_2^{\mathsf{auth}}))$, and that is what we needed. □

Secrecy. We note that secrecy of the exchanged key for STS should have a different formalization than the secrecy of the DH discussed in Section 8. In particular, we are interested in formalizing the property that if an honest agent completes a session ostensibly with another honest agent, then the shared key computed by the agent must be secret. This can again be modeled as an indistinguishability of two protocols Π_1^{sec} and Π_2^{sec}. In both protocols, we enhance the transition system for the STS protocol with an oracle query for revealing a shared key. In Π_1^{sec}, if the oracle query is to reveal the key in a completed session $\mathtt{id}_\alpha$ for A (B, respectively), then the oracle replies with the key computed by A (B, respectively) in that session. In Π_2^{sec}, if the oracle query is to reveal the key in a completed session $\mathtt{id}_\alpha$ for A (B, respectively), then the oracle checks if A (respectively) completed this session ostensibly with B (A, respectively): If it indeed is the case, then the oracle generated reveals a random key g^c; otherwise, it outputs the computed key. Π_1^{sec} and Π_2^{sec} can be easily shown to be indistinguishable as follows.

First, Π_1^{sec} can be easily transformed into a protocol, in which the oracle query before revealing A's (B's, respectively) computed secret in a completed session $\mathtt{id}_\alpha$ for A (B, respectively) checks if the other agent is ostensibly B (A, respectively). Whether the check succeeds or not, the oracle query still reveals the computed key. Now, in the branch when the aforementioned check passes, we can show in a manner similar to the proof of Proposition 10.1 that A (B, respectively) must be exchanging the secret with one of the two sessions of B (A, respectively) and the secret must be of the form $(g^b)^a$ ($(g^a)^b$, respectively), where a and b are the exponents in the corresponding sessions of A and B. Thus, the folded term representing the protocol Π_1^{sec} can be transformed into one in which the oracle query results in five different possible conditional outputs, four for the keys exchanged between the honest agents and one for all other cases. Similarly, the folded term representing the protocol Π_2^{sec} can also be transformed into one in which the oracle query results in five different possible conditional outputs, four of them being g^c when the two honest agents are involved in the exchange and one for all other cases. Now, these two terms can be proved to be indistinguishable using the computational DDH assumption along the same lines as the proof of $\Phi(\mathsf{fold}(\Pi_2')) \sim \Phi(\mathsf{fold}(\Pi_2))$ in the proof of Proposition 8.1.

11 FORMALIZING THE PROOFS IN COQ

We formalized the machine-checked proofs of the theorems, real-or-random secrecy of the Diffie-Hellman (DH) protocol, and authentication of the Station-to-Station (STS) protocol in Coq, an interactive theorem-prover [29]. In particular, we formalized the real-or-random secrecy of the DH protocol and for authentication of the STS protocol for one session each of responder and initiator. We did only one session to keep the size of the formulas small, but the idea for any fixed number of sessions is similar.

We also formalized the auxiliary theorems, for example, the derivation of the DDH assumption for three participants, and hence for any number of participants, using the assumption for two participants. All the machine-checked proofs are available at Reference [22].

There are three kinds of specifications that Coq supports, abstract types, mathematical collections, and logical propositions. These specifications are represented by sorts, `Type`, `Set`, and `Prop`, respectively.

11.1 Types

As presented in Section 2, the set of sorts S has at least two sorts message and bool. Using the feature of mutually inductive types [17] in Coq, we define message and bool types. The syntax is as follows:

```
Inductive message: Type :=
| Mvar: nat → message
| O: message
| N: nat → message
| ifm_then_else_: bool[3] → message → message → message
with bool: Type :=
| Bvar: nat → bool
| TRue: bool
| FAlse: bool
| eqm: message → message → bool
| eqb: bool → bool → bool
| ifb_then_else_: bool → bool → bool → bool.
```

As described in Reference [17], `ilist` is a type that takes length of the list as an argument and produces a length-indexed list. They define polymorphic length-indexed lists. Following Reference [17], we formalize the polymorphic length-indexed list below.

```
Inductive ilist (A:Type) : nat → Type :=
| Nil : ilist A 0
| Cons: forall n, A→ ilist A n → ilist A (S n).
```

Where the arguments `A` and `nat` represent type and length of the ilist, respectively. As described in Section 2, lists contain message, bool, or both. We needed a type that refers to either of the types. We call it `oursum`, and it is defined inductively on message and bool.

```
Inductive oursum : Type:=
| msg :  message → oursum
| bol :  bool  → oursum.
```

A frame is modeled as a length-indexed list of type `oursum`. This is declared as type `mylist n`, where `n` is length of the list.

```
Definition mylist : nat→ Type := ilist oursum.
```

11.2 Formalizing Indistinguishability Relation

A binary relation on a type `A` is formalized in Coq as

```
Definition relation (A:Type) := A → A → Prop.
```

[3]To avoid a clash with built-in type in Coq, `bool` is represented as `Bool`.

To increase the readability, we introduce few notations as follows:

```
(** ifm_then_else_ *)
Notation "'If' c1 'then' c2 'else' c3" := (ifm_then_else_ c1 c2 c3)
  (at level 200, right associativity,
  format "'[v  ' 'If' c1 '/' '[' 'then' c2 ']' '/' '[' 'else' c3 ']' ']'").
(** ifb_then_else_ *)
Notation "'IF' c1 'then' c2 'else' c3" := (ifb_then_else_ c1 c2 c3)
  (at level 200, right associativity,
  format "'[v  ' 'IF' c1 '/' '[' 'then' c2 ']' '/' '[' 'else' c3 ']' ']'").
(** pair *)
Notation "( x , y , .. , z )" := (pair .. (pair x y) ..  z).
```

Definition 11.1 (Copied Verbatim from Reference [29]). A parametric relation R is any term of type $\forall(x_1 : T_1) \ldots (x_n : T_n),\ relation\ A$. The expression A, which depends on $x_1 \ldots x_n$, is called the carrier of the relation and R is said to be a relation over A; the list $x_1, \ldots, x_n$ is the (possibly empty) list of parameters of the relation.

We model indistinguishability as a parametric relation on length-indexed lists of type `oursum`, `mylist n`, where the length `n` is a parameter of the relation. We write the indistinguishability relation as `EQI`, denoted as $\sim$. The formalization of `EQI` is achieved using the following command.

```
Parameter EQI: forall n, relation (mylist n).
```

An instance of a parametric relation `EQI` with a parameter is a term `(EQI n)` where `n` is a natural number, and models indistinguishabilty relation amongst frames of length `n`. The equivalence property of `EQI` is formalized as below.

```
Axiom EQI_equiv: forall n, equiv (mylist n) (@EQI n).
```

The axiom `EQI_equiv` states the fact that `EQI` is an equivalence relation. The properties reflexivity, symmetry, and transitivity of equivalence relation `EQI` are formalized in the theorems `EQI_ref`, `EQI_sym`, and `EQI_trans`, respectively. For example, the theorem `EQI_ref` is formalized as below.

```
Theorem EQI_ref: forall {n:nat} (ml: mylist n), ml ~ ml.
```

Similarly, other theorems are formalized. The parametric equivalence relation `EQI` can be declared with the following command.

```
Add Parametric Relation n : (mylist n) (@EQI n)
reflexivity proved by (EQI_ref )
symmetry proved by (EQI_sym )
transitivity proved by (EQI_trans )
as EQI_rel
```

Where the name `EQI_rel` uniquely identifies the relation and is used to generate fresh names, `EQI_rel_Reflexive`, `EQI_rel_Symmetric`, for automatically provided lemmas.

11.2.1 Formalization of =. Recall the abbreviation = defined in Section 2, which serves as a congruence. If we use the built-in equality of Coq to model this abbreviation together with its built-in axiomatic system, then we end up with an unsound extension of the Coq logic. This is because the Coq inequality forces two syntactically unequal terms of our logic to be unequal, which will contradict our axioms (for example, `IfTrue`). Instead, we model = as a pair of equivalence relations: `EQm` for modeling = amongst terms of type message and `EQb` for modeling = amongst terms of type bool.

The formalization of EQm and EQb is achieved with the following commands:

```
Definition EQm : relation message :=
                 fun (m1 m2: message) ⇒ [bol (eqm m1 m2)] ~ [ bol TRue].
Definition EQb : relation bool :=
                 fun (b1 b2: bool) ⇒ [bol (eqb b1 b2)] ~ [ bol TRue].
```

For the rest of the article, we write EQm as # and EQb as ##.

Even if we do not use the native =, we still would like to use native Coq tactics, which exploit the fact that Leibniz equality is a congruence. Thus, we declare EQm and EQb as morphisms, which allows us to exploit their congruence properties.

Definition 11.2 (Morphism (Copied Verbatim from Reference [29])). A parametric unary function f of type *forall* $(x_1 : T_1) \ldots (x_n : T_n)$, $A_1 \to A_2$ covariantly respects two parametric relation instances R_1 and R_2 if, whenever x, y satisfy $R_1\ x\ y$, their images $(f\ x)$ and $(f\ y)$ satisfy $R_2\ (f\ x)\ (f\ y)$. An f that respects its input and output relations is called a unary covariant *morphism*. The sequence $x_1, \ldots, x_n$ represents the parameters of the morphism.

Example 11.3. The constructor pair takes two terms of type message and gives a term of type message. We have the following axiom:

```
Axiom pair_Cong: forall (m1 m2 m1' m2': message), m1 # m1' → m2 # m2' → (m1, m2) # (m1', m2').
```

We add a morphism pair_mor of the function pair that respects the relation EQm as follows. The morphism pair_mor is declared using the following commands.

```
Add Parametric Morphism: (@ pair) with
  signature EQm ⇒ EQm ⇒ EQm as pair_mor.
 Proof. intros. apply pair_Cong; assumption. Qed.
```

The command declares pair as a morphism of the signature EQm ⇒ EQm ⇒ EQm. The identifier app_mor gives a unique name to the morphism. The command also requires us to prove that the function pair respects the relations identified by the signature. The proof uses the axiom pair_Cong. Similarly morphisms for other functions are formalized.

11.3 Axioms, Theorems, and Verification

We present the details of the formalization of the axioms that are required to prove the real-or-random secrecy of the DH protocol.

We formalize the axioms presented in the Table 1 and the DDH axiom. For example, we formalize the axiom IfSame as follows:

```
Axiom IFSAME_M: forall (b:Bool) (x : message), (If b then x else x) # x.
Axiom IFSAME_B: forall (b:Bool) (b1 : Bool),   (IF b then b1 else b1) ## b1.
```

Two axioms are used to distinguish the cases when x is a message and a bool. All other axioms in the Table 1 are similarly formalized.

The DDH axiom states that, for given messages g^{n_a} and g^{n_b}, where n_a, n_b, and n_c are freshly generated random numbers, the terms $g^{n_a n_b}$ and g^{n_c} are indistinguishable. We formalize the axiom as the following:

```
Axiom DDH : forall (n na nb nc: nat),  (Fresh [ n, na, nb, nc ] []) = true →
[ msg (G n), msg (g n), msg (n^^na), msg (n^^nb), msg (exp (G n) (n^^na) (r nb))]
  ~ [msg (G n), msg (g n), msg (n^^na), msg (n^^nb), msg (n^^nc)].
```

where we have leveraged the following the notation:

```
Notation "x '^^' y" := (exp (G x) (g x) (r y)).
```

The function Fresh captures the notion of freshly generated random numbers and each number in the list [n; na; nb; nc] is freshly generated. We also introduced a notation to compact the t.

Example 11.4. Using the axiom IFSAME_B, we proved the following theorem.

```
Theorem IFTF: forall (n:nat), (IF (Bvar n) then TRue else FAlse) ## (Bvar n).
Proof.
intros.
rewrite ← (IFSAME_B (Bvar n) (Bvar n)) at 2.
rewrite → IFEVAL_B with (b1 := (Bvar n)).
simpl.
rewrite ← beq_nat_refl.
reflexivity.
Qed.
```

where beq_nat_refl is a lemma that states, for any natural number n, true = (beq_nat n n).

As described in Section 8, real-or-random secrecy is modeled as indistinguishability of two protocols, Π_1, in which the real shared key is revealed, and Π_2, in which a randomly generated number is revealed. The formalization of this is achieved by proving that the two frames $\Phi(\text{fold}(\Pi_1))$ and $\Phi(\text{fold}(\Pi_2))$ are indistinguishable. The formalization of the frame $\Phi(\text{fold}(\Pi_1))$ is the following:

```
Definition phi10 := [ msg (G 0) ; msg (g 0)].
Definition phi11 := phi10 ++ [msg t10].
 ...
Definition phi15 := phi14 ++ [msg t14].
```

We use phi15 to represent the frame $\Phi(\text{fold}(\Pi_1))$ and it contains the list of messages available to attacker at the end of the protocol. The frame phi10 represents the initial knowledge of the attacker where as the frames phi11, phi12, and phi13 represent the attacker knowledge during the protocol execution. Of course, the symbol ++ stands for concatenation of two lists. Similarly the frame phi25 that represents $\Phi(\text{fold}(\Pi_2))$ is formalized. The following theorem illustrates the real-or-random secrecy of one matching session of the DH protocol.

```
Theorem Pi1_Pi2: phi15 ~ phi25.
```

We also formalized the authentication of the STS protocol using the axiom schema, existential unforgeability against chosen message attacks (UF-CMA), and the correctness axiom of digital signatures. The axioms UF-CMA and correctness are formalized below:

```
Section ds_axioms.
(** Digital Signatures*)
(** Correctness *)
Axiom correctness : forall (n:nat) (t t': message), (ver (pk n) t (sign (sk n) t t')) ## TRue.
(** Existential unforgeability against chosen message attacks (UFCMA) *)
Fixpoint unforgb (j:nat) (n:nat) (ml: list message) (t u :message): Bool :=
  match j, ml with
    | 0, _ ⇒ FAlse
    | S _, [] ⇒ FAlse
    | S j', h:: tl ⇒ match h with
                       | (sign (sk n) t1 t2) ⇒ IF (eqm t t1) then (ver (pk n) t1 u)
                                                             else (unforgb j' n tl t u)
                       | _ ⇒ FAlse
                     end
  end.
```

```
Axiom UFCMA : forall (n : nat)( t u: message), (closMylist [msg t, msg u] = true) ∧
      (insec_n_mylis n [msg t, msg u] = false) →
      let j := length(list_skn_in_sign n ((subtrmls_msg t) ++(subtrmls_msg u))) in
      let ml := distsigntrms n ((subtrmls_msg t) ++ ( subtrmls_msg u)) in
      (ver (pk n) t u) ## (unforgb j n ml t u).
End ds_axioms.
```

The authentication property of the STS protocol is modeled as proving the two frames $\Phi(\text{fold}(\Pi_1^{\text{auth}}))$ and $\Phi(\text{fold}(\Pi_2^{\text{auth}}))$ are equal. The frame $\Phi(\text{fold}(\Pi_1^{\text{auth}}))$ is formalized as below:

```
Definition sphi10 := [msg (G n); msg (g n); msg A; msg (pk nA); msg B; msg (pk nB); msg Q1;
                      msg (pk c1); msg Q2; msg (pk c2)].
Definition sphi11 := sphi10 ++ [msg s10].
Definition sphi12 := sphi11 ++ [msg s11].
Definition sphi13 := sphi12 ++ [msg s12].
Definition sphi14 := sphi13 ++ [msg s13].
Definition sphi15 := sphi14 ++ [msg s14].
```

The frame sphi15 represents the frame $\Phi(\text{fold}(\Pi_1^{\text{auth}}))$ and it contains the messages that are available to the attacker at the end of the protocol. sphi10 represents the initial frame, sphi11, sphi12, sphi13, and sphi14 represent the attacker knowledge during execution of the protocol. Similarly, the frame sphi25 that represents $\Phi(\text{fold}(\Pi_2^{\text{auth}}))$ is formalized.

The authentication property of STS protocol is illustrated by the following theorem.

```
Theorem IND_DH_AUTH: sphi15 = sphi25.
```

Where the symbol = represents the equality of the two frames sphi15 and sphi25.

12 SECURITY OF ENCRYPTIONS AND FURTHER VERIFICATION RESULTS

We shall now show that the standard IND-CPA, IND-CCA1 and IND-CCA2 security notions for encryption (see, e.g., Reference [13]) can be easily translated to the BC framework, illustrating the convenience of the BC framework. We assume the function symbol k and abbreviations sk, pk are as in Section 9.

12.1 Encryptions

Let $\{\_\}^{\_}_{\_} : \text{message} \times \text{message} \times \text{message} \to \text{message}$ and $\text{dec}(\_,\_) : \text{message} \times \text{message} \to \text{message}$ and $\text{r}(\_) : \text{message} \to \text{message}$ be function symbols for encryption, decryption and random seed generation satisfying $\text{dec}(\{x\}^z_{\text{pk}(y)}, \text{sk}(y)) = x$. (r is not to be confused with r we used for group exponentiation.) Let $\text{L} : \text{message} \to \text{message}$ be a function symbol for length such that $[\![\text{L}]\!](d)(w;\rho_1;\rho_2) := 1^{|d(w;\rho_1;\rho_2)|}$, where for a bit string s, $|s|$ denotes its length. Let $\vec{t}\,[x]$ be a list of terms with a single variable x. For a closed term v, let $\vec{t}\,[v]$ denote the term that we obtain from $\vec{t}\,[x]$ by replacing all occurrences of x by v. Let u, u', u'' be closed terms. Consider the formula

$$\vec{t}\left[\text{if EQ}(\text{L}(u),\text{L}(u')) \text{ then } \{u\}^{\text{r}(n_2)}_{\text{pk}(n_1)} \text{ else } u''\right] \sim \vec{t}\left[\text{if EQ}(\text{L}(u),\text{L}(u')) \text{ then } \{u'\}^{\text{r}(n_3)}_{\text{pk}(n_1)} \text{ else } u''\right],$$

in which $n_1 \in \mathcal{N}$ occurs only as $\text{k}(n_1)$, $\text{sk}(n_1)$ only occurs in decryption position (that is, as in $\text{dec}(\_,\text{sk}(n_1))$), and n_2, n_3 do not occur anywhere else. We call the above formula

- Enc_{CPA} if $\text{sk}(n_1)$ does not occur anywhere,
- Enc_{CCA1} if for any $t'[x]$ subterm of $\vec{t}\,[x]$ with x explicitly occurring in $t'[x]$, the term $\text{dec}(t'[x],\text{sk}(n_1))$ is not a subterm of $\vec{t}\,[x]$, and

- $\mathsf{Enc}_{\mathrm{CCA2}}$ if for any $t'[x]$ term with x explicitly occurring in $t'[x]$, the term $\mathsf{dec}(t'[x], \mathsf{sk}(n_1))$ occurs only as

$$\text{if } \mathsf{EQ}(t'[x], x) \wedge \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \text{ then } t''[x] \text{ else } \mathsf{dec}(t'[x], \mathsf{sk}(n_1)).$$

  Formally, the formula above is $\mathsf{Enc}_{\mathrm{CCA2}}$ if each component term $t_i[x]$ of the vector $\vec{t}[x]$ is $(n_1, u, u') - \mathsf{Enc}_{CCA2}$ compliant as defined (recursively) below. We say that a term $t[x]$ is $(n, u, u') - \mathsf{Enc}_{CCA2}$ *compliant* if one of the following holds:
  — $t[x]$ is a ground term, not equal n.
  — $t[x] \equiv \mathsf{pk}(n)$.
  — $t[x]$ is the variable x.
  — There is a function symbol $f \in \mathcal{F} \cup \mathcal{G}$ and terms $t^1[x], \ldots, t^r[x]$ such that $t[x] \equiv f(t^1[x], t^2[x], \ldots, t^r[x])$ and for any $t'[x]$ term containing x, $t[x] \not\equiv \mathsf{dec}(t'[x], \mathsf{sk}(n))$ and $t^i[x]$ is $(n, u, u') - \mathsf{Enc}_{CCA2}$ *compliant* for each $i = 1, \ldots, r$.
  — There are $(n, u, u') - \mathsf{Enc}_{CCA2}$ *compliant* terms $t'[x], t''[x]$, such that $t[x]$ is

$$\text{if } \mathsf{EQ}(t'[x], x) \wedge \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \text{ then } t''[x] \text{ else } \mathsf{dec}(t'[x], \mathsf{sk}(n)).$$

Intuitively, x represents the place for the left-right encryption oracle response in the security game for encryption. Terms that can be computed before using the left-right encryption oracle are those that do not contain x. As CPA security does not allow decryption oracle, we allow no decryption. CCA1 allows decryption request before the encryption request, hence decryption can be applied to terms without x. In CCA2, we have to make sure that if decryption is applied on $t'[x]$ term containing x, then $t'[x]$ is not the encryption oracle response, namely, x, and if it is, then the decryption returns $\mathbf{0}$, and some $t''[x] \not\equiv \mathsf{dec}(t'[x], \mathsf{sk}(n_1))$ is used. In fact, this definition of $\mathsf{Enc}_{\mathrm{CCA1}}$ is equivalent with the one in Reference [9].

Theorem 12.1. *If* $[\![\mathsf{k}(\_), \{\_\}_{\_}, \mathsf{dec}(\_,\_)]\!]$ *is CPA secure, then* Enc_{CPA} *is computationally sound. If it is CCA1 secure, then* Enc_{CCA1} *is computationally sound. If it is CCA2 secure, then* Enc_{CCA2} *is computationally sound. Conversely, if there is a constant* $\ell \in \mathbb{N}$ *and a CPA (or CCA1 or CCA2, respectively) attack* $\mathcal{A}$ *against* $[\![\mathsf{k}(\_), \{\_\}_{\_}, \mathsf{dec}(\_,\_)]\!]$ *such that the number of oracle queries* $\mathcal{A}$ *makes does not exceed* ℓ *for any* η*, then* Enc_{CPA} *(or* Enc_{CCA1} *or* Enc_{CCA2}*, respectively) axiom is violated in some computational model with the given interpretation* $[\![\mathsf{k}(\_), \{\_\}_{\_}, \mathsf{dec}(\_,\_)]\!]$.

Proof. We prove the validity of $\mathsf{Enc}_{\mathrm{CCA2}}$, the others are analogous but simpler. We proceed by contradiction. Assume that there is a list of terms $\vec{t}[x]$ with a single variable x, and closed terms u, u', u'' as well as names n_1, n_2, n_3, and a computational model $\mathcal{M}^c$ such that

$$\mathcal{M}^c \not\models \vec{t}\left[\begin{array}{r}\text{if } \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \text{ then } \{u\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \\ \text{else } u''\end{array}\right] \sim \vec{t}\left[\begin{array}{r}\text{if } \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \text{ then } \{u'\}^{\mathsf{r}(n_3)}_{\mathsf{pk}(n_1)} \\ \text{else } u''\end{array}\right].$$

This means that there is a Turing machine $\mathcal{A}$ that runs in polynomial time in the security parameter η such that

$$\begin{aligned} Adv^{\mathcal{A}}(\eta) = |\mathbf{Prob}\{\rho :\ & \mathcal{A}([\![\vec{t}\,[\text{if } \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \text{ then } \{u\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \text{ else } u'']\!]_{\rho,\eta}; \rho_2) = 1\} - \\ & \mathbf{Prob}\{\rho :\ \mathcal{A}([\![\vec{t}\,[\text{if } \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \text{ then } \{u'\}^{\mathsf{r}(n_3)}_{\mathsf{pk}(n_1)} \text{ else } u'']\!]_{\rho,\eta}; \rho_2) = 1\}| \end{aligned}$$

is a non-negligible function in η. Then an adversary $\mathcal{B}$ can win the IND-CCA2 game against $\mathsf{k}(n_1)$ as follows. As usual in the IND-CCA2 security definition, on the input 1^η, the encryption oracle first generates an internal bit b randomly, and a public-key secret-key pair (pk, sk). $\mathcal{B}$ is given pk by the oracle. $\mathcal{B}$ generates bit strings for the names that occur in $\vec{t}[x], u, u', u''$ (except for n_1, n_2, n_3) according to the way names are generated in $\mathcal{M}^c$. Using these, $\mathcal{B}$ then computes the interpretations of u, u', u'', and subterms of $\vec{t}[x]$ that do not contain x: the only thing $\mathcal{B}$ does not

have access to in these terms is the interpretation sk of $\mathsf{sk}(n_1)$, but according to our assumption, that only occurs in decryption positions. So those interpretations are computed by submitting the interpretation in the cyphertext position to the decryption oracle. (The case of $\mathsf{Enc}_{\mathrm{CCA1}}$ is the same, while the $\mathsf{Enc}_{\mathrm{CPA}}$ axiom does not allow $\mathsf{sk}(n_1)$ to occur at all.) Once all these terms are computed, if the interpretation of u and the interpretation of u' have the same length, then $\mathcal{B}$ submits the two interpretations to the encryption oracle, which then returns the encryption of one of them, c. The oracle generates the interpretation of n_2 or n_3 depending on which plaintext it encrypts. If the two lengths disagree, then let us call c the interpretation of u''. Then $\mathcal{B}$ continues computing all of the interpretation of $\vec{t}[x]$ by substituting c for x. This computation in the $\mathsf{Enc}_{\mathrm{CCA2}}$ axiom may again contain decryptions by $\mathsf{sk}(n_1)$, but as they are assumed to be guarded by the assumptions that those decrypted terms are not equal to the value returned by the encryption oracle, again submission to the decryption oracle is possible. (In $\mathsf{Enc}_{\mathrm{CCA1}}$ and $\mathsf{Enc}_{\mathrm{CPA}}$ cases here $\mathsf{sk}(n_1)$ is not allowed to occur any more.) When $\mathcal{B}$ finishes the computation of the interpretation of $\vec{t}(x)$, it hands the result over to $\mathcal{A}$. When $\mathcal{A}$ finishes, $\mathcal{B}$ outputs the output of $\mathcal{A}$. By the construction of $\mathcal{B}$,

$$\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 0\} = \mathbf{Prob}\{\rho : \ \mathcal{A}([\![\vec{t} \begin{bmatrix} \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \\ \mathsf{then}\ \{u\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \\ \mathsf{else}\ u'' \end{bmatrix}]\!]_{\rho,\eta}; \rho_2) = 1\} \text{ and}$$

$$\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 1\} = \mathbf{Prob}\{\rho : \ \mathcal{A}([\![\vec{t} \begin{bmatrix} \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u')) \\ \mathsf{then}\ \{u'\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \\ \mathsf{else}\ u'' \end{bmatrix}]\!]_{\rho,\eta}; \rho_2) = 1\}.$$

Thus, the quantity $|\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 1\} - \mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 0\})|$ is non-negligible by our assumption. Now,

$$\begin{aligned} |\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = b\} - \tfrac{1}{2}| &= |\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 1\} + \mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 0 \ \wedge \ b = 0\} - \tfrac{1}{2}| \\ &= |\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 1\} + (\tfrac{1}{2} - \mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 0\}) - \tfrac{1}{2}| \\ &= |\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 1\} - \mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 0\})|. \end{aligned}$$

As the quantity $|\mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 1\} - \mathbf{Prob}\{\mathcal{B}(1^\eta, pk) = 1 \ \wedge \ b = 0\})|$ is non-negligible, IND-CCA2 security is broken by $\mathcal{B}$.

The proof of the converse is the following. Let us consider an IND-CCA2 attacker $\mathcal{A}$ on the given interpretation of $\mathsf{k}(\_)$, $\{\_\}^{\_}_{\_}$, $\mathsf{dec}(\_,\_)$ that succeeds with non-negligible probability and makes at most ℓ oracle queries. Note that the actual number of oracle requests may vary by η and ρ, but we can always add requests the answers of which are ignored, so without loss of generality, we can assume that there are uniformly ℓ submissions, of which the mth is the submission to the encryption oracle.

Fix a name n and function symbols $f_0, f_1, \ldots, f_{m-1}, f_m, \ f'_m, \ f''_m, \ f_{m+1}, \ \ldots, f_\ell \in \mathcal{G}$. Let $t_1, t_2, \ldots, t_{m-1}, u, u', u'', t_{m+1}[x], \ldots, t_\ell[x]$ be defined as follows:

$$\begin{aligned} t_1 &\stackrel{\mathrm{def}}{\equiv} f_0(\mathsf{pk}(n)) \\ t_{i+1} &\stackrel{\mathrm{def}}{\equiv} f_{i+1}(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), \ldots, \mathsf{dec}(t_i, \mathsf{sk}(n))) \\ u &\stackrel{\mathrm{def}}{\equiv} f_m(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), \ldots, \mathsf{dec}(t_{m-1}, \mathsf{sk}(n))) \\ u' &\stackrel{\mathrm{def}}{\equiv} f'_m(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), \ldots, \mathsf{dec}(t_{m-1}, \mathsf{sk}(n))) \end{aligned}$$

$$\begin{aligned} u'' &\stackrel{\text{def}}{\equiv} f''_m(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), \ldots, \mathsf{dec}(t_{m-1}, \mathsf{sk}(n))) \\ t_{j+1}[x] &\stackrel{\text{def}}{\equiv} f_{j+1}(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), \ldots, x, \ldots, c_j) \\ \vec{t}[x] &\stackrel{\text{def}}{\equiv} t_\ell[x], \end{aligned}$$

where for $j = m, \ldots, \ell - 1$,

$$c_j = \mathsf{if}\ \mathsf{EQ}(t_j[x], x) \wedge \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathsf{dec}(t_j[x], \mathsf{sk}(n_1)).$$

Let $\mathcal{M}^c$ be a model with the following the interpretations of the function symbols $f_0, \ldots, f_\ell$.

$[\![f_0]\!]$ is the Turing Machine that on input s_0 and tapes $\rho_1; \rho_2$ simulates the attacker $\mathcal{A}$ until it prepares a message query to be submitted to the decryption oracle. At that point $[\![f_0]\!]$ outputs the actual query and stops. For $0 < i < m$, $[\![f_i]\!]$ is the Turing Machine that on input s_0, s_1, s_2, s_{i-1} simulates the attacker $\mathcal{A}$ until it prepares the ith query to be submitted to the decryption oracle. Let s_i be the response of the decryption oracle. $[\![f_m]\!]$, $[\![f'_m]\!]$, and $[\![f''_m]\!]$ simulate $\mathcal{A}$ until $\mathcal{A}$ is ready to output the pair of messages to the encryption oracle. $[\![f_m]\!]$ outputs the first, $[\![f'_m]\!]$ outputs the second of the pair to be submitted to the encryption oracle if they have the same length, while the output of $[\![f''_m]\!]$ is used in further computations if their lengths differ. $[\![f_j]\!]$ for $j > m$ is similar, $[\![f_j]\!]$ is the Turing Machine that on input $s_0, s_1, s_2, s_{m-1}, c, s_{m+1}, \ldots, s_{j-1}$ simulates the attacker $\mathcal{A}$ until it prepares the jth query s_j to be submitted to the decryption oracle. Here, c is what the encryption oracle returns, while s's are what the decryption oracle returned. We claim that with these definitions,

$$\mathcal{M}^c \not\models \vec{t}\begin{bmatrix} \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \{u\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \\ \mathsf{else}\ u'' \end{bmatrix} \sim \vec{t}\begin{bmatrix} \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \{u'\}^{\mathsf{r}(n_3)}_{\mathsf{pk}(n_1)} \\ \mathsf{else}\ u'' \end{bmatrix}.$$

Let $\mathcal{B}$ be the algorithm that simply outputs its first input. Then,

$$\mathbf{Prob}\{\rho :\ \mathcal{B}([\![\vec{t}\begin{bmatrix} \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \{u\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \\ \mathsf{else}\ u'' \end{bmatrix}]\!]_{\rho,\eta}; \rho_2) = 1\} = \mathbf{Prob}\{\mathcal{A}(1^\eta, pk) = 1\ \wedge\ b = 0\}\ \text{and}$$

$$\mathbf{Prob}\{\rho :\ \mathcal{B}([\![\vec{t}\begin{bmatrix} \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \{u'\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)} \\ \mathsf{else}\ u'' \end{bmatrix}]\!]_{\rho,\eta}; \rho_2) = 1\} = \mathbf{Prob}\{\mathcal{A}(1^\eta, pk) = 1\ \wedge\ b = 1\}.$$

But just as before,

$$|\mathbf{Prob}\{\mathcal{A}(1^\eta, pk) = 1\ \wedge\ b = 1\} - \mathbf{Prob}\{\mathcal{A}(1^\eta, pk) = 1\ \wedge\ b = 0\})| = |\mathbf{Prob}\{\mathcal{A}(1^\eta, pk) = b\} - \tfrac{1}{2}|.$$

According to our assumption, $\mathcal{A}$ violates IND-CCA2 security, hence $|\mathbf{Prob}\{\mathcal{A}(1^\eta, pk) = b\} - \frac{1}{2}|$ is non-negligible, hence so is

$$\begin{aligned} &|\mathbf{Prob}\{\rho :\ \mathcal{B}([\![\vec{t}[\mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \{u\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)}\ \mathsf{else}\ u'']]\!]_{\rho,\eta}; \rho_2) = 1\} - \\ &\mathbf{Prob}\{\rho :\ \mathcal{B}([\![\vec{t}[\mathsf{if}\ \mathsf{EQ}(\mathsf{L}(u), \mathsf{L}(u'))\ \mathsf{then}\ \{u'\}^{\mathsf{r}(n_2)}_{\mathsf{pk}(n_1)}\ \mathsf{else}\ u'']]\!]_{\rho,\eta}; \rho_2) = 1\}|, \end{aligned}$$

and that completes the proof. □

Example 12.2. Suppose that nonce and key generation are such that there are 0-ary function symbols ℓ_{nonce} and ℓ_{skey} such that $\mathsf{L}(n) = \ell_{\text{nonce}}$ and $\mathsf{L}(\mathsf{sk}(n)) = \ell_{\text{skey}}$, and suppose also that pairing is length regular, that is, $\mathsf{L}(x_1) = \mathsf{L}(x_2) \wedge \mathsf{L}(y_1) = \mathsf{L}(y_2) \rightarrow \mathsf{L}(\langle x_1, x_2\rangle) = \mathsf{L}(\langle y_1, y_2\rangle)$. Note also that

from the definition of the interpretation of L, the formula $\mathsf{L}(\mathsf{L}(x)) = \mathsf{L}(x)$ is sound. Consider real-or-random secrecy of n (let $\mathsf{k}_i \equiv \mathsf{k}(n_i)$, $\mathsf{r}_i \equiv \mathsf{r}(n_{i+2})$):

$$\begin{aligned} &\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\right)\right), n\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n \\ &\sim \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\right)\right), n\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n', \end{aligned} \tag{1}$$

with $f \in \mathcal{G}$. It is easy to show that the core axioms together with $\mathsf{Enc}_{\mathrm{CCA2}}$ and the above properties of L, and the equations for pairing–projections, encryption–decryption imply this formula. The intuition of course is that sk_1 is hidden by the encryption with pk_2, the decrypted message in the second encryption is n_5, hence no key cycle occurs encrypting with pk_1, and so the second encryption does not reveal information about n. The key point of the proof is to transform first the terms so that $\mathsf{Enc}_{\mathrm{CCA2}}$ can be applied. For example, since dec acts on $f(\ldots)$, we have to make sure that there is a conditioning as we required in the definition of $\mathsf{Enc}_{\mathrm{CCA2}}$. So, we start the proof by rewriting $f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})$ according to

$$f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}) = \mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\ \mathsf{then}\ f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\ \mathsf{else}\ f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})$$

by `IfSame`, and then applying `IfMorph` and the equations for encryption and pairing, we obtain

$$\begin{aligned} &\pi_2(\mathsf{dec}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2)) \overset{\mathsf{EqCong}}{=} \\ &\pi_2\left(\mathsf{dec}\left(\mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\ \mathsf{then}\ f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\ \mathsf{else}\ f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\right)\right) \overset{\text{Ex. 7.3}}{=} \\ &\pi_2\left(\mathsf{dec}\left(\mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\ \mathsf{then}\ \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\ \mathsf{else}\ f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\right)\right) \overset{\mathsf{IfMorph}}{=} \end{aligned}$$

$$\begin{aligned} &\mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}) && \mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}) \\ &\mathsf{then}\ \pi_2(\mathsf{dec}(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \mathsf{sk}_2)) && = \mathsf{then}\ n_5 \\ &\mathsf{else}\ \pi_2(\mathsf{dec}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2)) && \mathsf{else}\ \pi_2(\mathsf{dec}(f(\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2)). \end{aligned}$$

Let us define $\vec{t}[x] := x, \{\mathsf{if}\ \mathsf{EQ}(f(x), x)\ \mathsf{then}\ n_5\ \mathsf{else}\ \pi_2(\mathsf{dec}(f(x), \mathsf{sk}_2)), n\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n$. Note that $\vec{t}$ for k_2 satisfies the conditions for $\mathsf{Enc}_{\mathrm{CCA2}}$, because the only decryption term (with the decryption key sk_2) containing x is $\mathsf{dec}(f(x), \mathsf{sk}_2))$, but this term occurs only under

$$\mathsf{if}\ \mathsf{EQ}(f(x), x)\ \mathsf{then}\ n_5\ \mathsf{else}\ \pi_2(\mathsf{dec}(f(x), \mathsf{sk}_2)),$$

$f(x)$ corresponding to $t'[x]$ in the definition of $\mathsf{Enc}_{\mathrm{CCA2}}$ and n_5 corresponding to t''. Now let $\vec{u}$ be the same list of terms as $\vec{t}$ except that it ends with n' instead of n. Hence, $\vec{u}$ similarly satisfies the conditions of $\mathsf{Enc}_{\mathrm{CCA2}}$. Note then that the Equation (1) is the same as

$$\vec{t}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \sim \vec{u}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right].$$

Note also that because of our assumptions on the length (at the beginning of this example), for a fresh n_6,

$$\mathsf{L}(\langle \mathsf{sk}_1, n_5\rangle) = \mathsf{L}(\langle \mathsf{sk}(n_6), n_5\rangle).$$

By the definition of = and Example 7.2,

$$\mathsf{EQ}(\mathsf{L}(\langle \mathsf{sk}_1, n_5\rangle), \mathsf{L}(\langle \mathsf{sk}(n_6), n_5\rangle)) = \boldsymbol{true},$$

and so by `IfTrue`,

$$\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2} = \mathsf{if}\ \mathsf{EQ}(\mathsf{L}(\langle \mathsf{sk}_1, n_5\rangle), \mathsf{L}(\langle \mathsf{sk}(n_6), n_5\rangle))\ \mathsf{then}\ \{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\ \mathsf{else}\ \mathbf{0}$$

and

$$\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2} = \text{if } \mathsf{EQ}(\mathsf{L}(\langle \mathsf{sk}_1, n_5\rangle), \mathsf{L}(\langle \mathsf{sk}(n_6), n_5\rangle)) \text{ then } \{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2} \text{ else } \mathbf{0}.$$

Using this, EqCong and $\mathsf{Enc}_{\mathsf{CCA2}}$, we get that

$$\vec{t}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \sim \vec{t}\left[\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \tag{2}$$

and

$$\vec{u}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \sim \vec{u}\left[\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right]. \tag{3}$$

With these moves, we have removed sk_1 from under the encryptions in Equation (1). Now,

$$\vec{t}\left[\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] = \{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\right)\right), n\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n. \tag{4}$$

And, again, by the length assumptions, for a fresh nonce n_7,

$$\mathsf{L}\left(\left\langle \pi_2\left(\mathsf{dec}\left(f\left(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right), \mathsf{sk}_2\right)\right), n\right\rangle\right) = \mathsf{L}\left(\left\langle \pi_2\left(\mathsf{dec}\left(f\left(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right), \mathsf{sk}_2\right)\right), n_7\right\rangle\right).$$

Hence, applying $\mathsf{Enc}_{\mathsf{CCA2}}$ for a second time just as before, but now for k_1, we obtain that

$$\begin{gathered}\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f\left(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right), \mathsf{sk}_2\right)\right), n\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n, \\ \sim \\ \{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f\left(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right), \mathsf{sk}_2\right)\right), n_7\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n.\end{gathered} \tag{5}$$

Putting together Equations (2), (4), and (5), with Trans and EqCong, we have that

$$\vec{t}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \sim \{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f\left(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right), \mathsf{sk}_2\right)\right), n_7\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n. \tag{6}$$

The same way we can derive that

$$\vec{u}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \sim \{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \left\{\pi_2\left(\mathsf{dec}\left(f\left(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right), \mathsf{sk}_2\right)\right), n_7\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n'. \tag{7}$$

But the right-hand sides of Equations (6) and (7) are equivalent as an immediate consequence of axioms FreshInd and Restr. Finally, again transitivity delivers

$$\vec{t}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right] \sim \vec{u}\left[\{\mathsf{sk}_1, n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\right],$$

which is what we wanted to show.

The axioms for IND-CPA, IND-CCA1, IND-CCA2 and the proof of the above example have also been mechanized in Coq.

Remark 5. Example 12.2 illustrates the advantage of the BC technique for indistinguishability over the BC technique for reachability [8] for CCA2 encryption. In the BC technique for reachability, the proof that n cannot be computed from the two encrypted messages can be only done with the complicated key-usability notion in Reference [10]. Here, we could simply use $\mathsf{Enc}_{\mathsf{CCA2}}$ and no new predicate was needed.

13 CONCLUSIONS

We have introduced key extensions to the core of computationally complete symbolic attacker based on indistinguishability first introduced in Reference [9] that are necessary to apply the technique to analyze protocols allowing multiple sessions. Toward this end, we introduced a number of new axioms for the if _ then _ else _ function symbol, a core element of the technique. We have illustrated how these axioms work through several small examples. We also introduced axioms expressing DDH assumption, UF-CMA unforgeability of signatures, IND-CPA, IND-CCA1, and IND-CCA2 security of encryptions that are immediate translations of the corresponding computational properties to the framework. Through the verification of real-or-random secrecy of the DH key exchange protocol and the verification of authentication of a simplified version of the STS protocol, we showed how the model can be used to tackle multiple sessions, algebraic properties, real-or-random secrecy, and even trace properties. The axiomatic system and the proofs of auxiliary theorems and security properties of protocols have been mechanized in Coq.

One direction that we plan to investigate is to extend the Computationally Complete Symbolic Attacker technique to deal with an arbitrary fixed number of sessions. For this, we intend to formalize induction in the Computationally Complete Symbolic Attacker technique. Other directions of future work are decidability results and automation. We believe that our logic is undecidable in general, but tractable for verification of interesting class of protocols. The latter belief is based on the procedures and techniques designed in Reference [26] for verification of reachability properties in the BC framework. For reachability, verification for a large classes of protocols turns out to be decidable in co-NP [26]. Finally, we also plan to investigate extending the technique to reason about a polynomial number of sessions.

ACKNOWLEDGMENTS

We are indebted to Hubert Comon-Lundh and Adrien Koutsos for the invaluable discussions. We also thank anonymous reviewers who have provided useful comments. Part of the research was carried out while Gergei Bana was at INRIA Paris and then at the University of Luxembourg.

REFERENCES

[1] M. Abadi and C. Fournet. 2001. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*. ACM, 104–115.

[2] M. Abadi and P. Rogaway. 2002. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptol.* 15, 2, 103–127.

[3] M. Abdalla, P.-A. Fouque, and D. Pointcheval. 2005. Password-based authenticated key exchange in the three-party setting. In *Proceedings of the 8th International Conference on Theory and Practice in Public Key Cryptography (PKC'05)*. Springer, 65–84.

[4] M. Backes, B. Pfitzmann, and M. Waidner. 2003. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*. ACM, 220–230.

[5] G. Bana, P. Adão, and H. Sakurada. 2012. Computationally complete symbolic attacker in action. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12) (LIPIcs)*. Schloss Dagstuhl, 546–560.

[6] G. Bana and R. Chadha. 2016. Verification Methods for the Computationally Complete Symbolic Attacker Based on Indistinguishability. Retrieved from http://eprint.iacr.org/2016/069.

[7] G. Bana, R. Chadha, and A. K. Eeralla. 2018. Formal analysis of vote privacy using computationally complete symbolic attacker. In *Proceedings of the 23rd European Symposium on Research in Computer Security (ESORICS'18)*. Springer, 350–372.

[8] G. Bana and H. Comon-Lundh. 2012. Towards unconditional soundness: Computationally complete symbolic attacker. In *Proceedings of the 1st International Conference on Principles of Security and Trust (POST'12)*. Springer, 189–208.

[9] G. Bana and H. Comon-Lundh. 2014. A computationally complete symbolic attacker for equivalence properties. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. ACM, 609–620.

[10] G. Bana, K. Hasebe, and M. Okada. 2013. Computationally complete symbolic attacker and key exchange. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. ACM, 1231–1246.
[11] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. 2015. Mind the Gap: Modular Machine-checked Proofs of One-Round Key Exchange Protocols. Retrieved from http://eprint.iacr.org/.
[12] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. 2011. Computer-aided security proofs for the working cryptographer. In *Proceedings of the 31st Annual Conference on Advances in Cryptology (CRYPTO'11)*. Springer, 71–90.
[13] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. 1998. Relations among notions of security for public-key encryption schemes. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'98)*. Springer, 26–45.
[14] B. Blanchet. 2005. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *Proceedings of the 20th International Conference on Automated Deduction (CADE'05)*. Springer.
[15] B. Blanchet. 2008. A computationally sound mechanized prover for security protocols. *IEEE Trans. Depend. Secure Comput.* 5, 4 (2008), 193–207.
[16] D. Boneh. 1998. The decision diffie-hellman problem. In *Proceedings of the Conference on Algorithmic Number Theory (ANTS'98)*. Springer, 48–63.
[17] Adam Chlipala. 2013. *Certified Programming with Dependent Types—A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press. I–XII, 1–424 pages. Retrieved from http://mitpress.mit.edu/books/certified-programming-dependent-types.
[18] H. Comon and A. Koutsos. 2017. Formal computational unlinkability proofs of RFID protocols. In *Proceedings of the IEEE 30th Computer Security Foundations Symposium (CSF'17)*. 100–114.
[19] H. Comon-Lundh and V. Cortier. 2008. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, 109–118.
[20] V. Cortier, C. C. Dragan, F. Dupressoir, B. Schmidt, P. Strub, and B. Warinschi. 2017. Machine-checked proofs of privacy for electronic voting protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*. 993–1008.
[21] C. Cremers. 2008. The scyther tool: Verification, falsification, and analysis of security protocols. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08)*, Vol. 5123. Springer, 414–418.
[22] Ajay Kumar Eeralla. 2019. Coq formalization of Computationally Complete Symbolic Attacker. Retrieved from https://bitbucket.org/ajayeeralla/machine-checked-proofs/src/master/.
[23] P. Gupta and V. Shmatikov. 2005. Towards computationally sound symbolic analysis of key exchange protocols. In *Proceedings of the ACM Workshop on Formal Methods in Security Engineering (FMSE'05)*. ACM, 23–32.
[24] J. Katz and Y. Lindell. 2007. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press.
[25] R. Küsters and M. Tuengerthal. 2009. Computational soundness for key exchange protocols with symmetric encryption. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'09)*. ACM, 91–100.
[26] Guillaume Scerri. 2015. *Proofs of security protocols revisited*. Thèse de doctorat. Laboratoire Spécification et Vérification, ENS Cachan, France. Retrieved from http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/scerri-phd15.pdf.
[27] G. Scerri and S.-O. Ryan. 2016. Analysis of key wrapping APIs: Generic policies, computational security. In *Proceedings of the IEEE 29th Computer Security Foundations Symposium (CSF'16)*. IEEE Computer Society, 281–295.
[28] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin. 2012. Automated analysis of diffie-hellman protocols and advanced security properties. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF'12)*. 78–94.
[29] Yves Bertot and Pierre Castéran. 2004. A brief overview. In *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer Berlin Heidelberg, 1–11. https://doi.org/10.1007/978-3-662-07964-5_1

Received September 2017; revised May 2019; accepted July 2019