Verification Methods for the Computationally Complete Symbolic Attacker Based on Indistinguishability

Gergei Bana
INRIA Paris-Rocquencourt
Paris, France
bana@math.upenn.edu

Rohit Chadha
University of Missouri
Colombia MO, United States
chadhar@missouri.edu

ABSTRACT

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) [6]. In this paper we argue that the real breakthrough of this technique is the recent introduction of its version for indistinguishability [7] 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.

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 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 [12], SCYTHER [17].

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 [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 [13] and EASYCRYPT [10]. Both of these approaches have limitations. Computational soundness results require too 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, often the current state-of-the-art formal provers are 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 in [6] (and developed in [5, 8, 22, 7]), formalizes protocols and attackers in first-order logic. The notion of *symbolic attacker* is kept, but instead of specifying a restricted list of actions that an attacker can do as is the case in the Dolev-Yao model, [6] only specifies facts that the attacker cannot violate. 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. 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 paper, we shall call this approach the BC technique.

When verifying security of protocols with the BC technique, one tries to prove that the negation of the security goal expressed as a first-order formula is logically inconsistent with the axioms. If one manages to establish inconsistency, then a proof of security in the computational model follows, as in this approach the attacker is allowed to perform all operations that a computational attacker can do. Failure to prove this inconsistency yields a first-order model consisting of a sequence of symbolic actions, which a computational attacker may perform to launch an attack (e.g. in [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 [6] have coined the term *computationally complete symbolic attacker* for the symbolic attacker in this approach.

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 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 a secure protocol. Furthermore, this approach is not stricken by the *commitment problem* (see e.g. [21]), an issue with all other symbolic verification techniques.

While the initial papers on the computationally complete symbolic attacker focussed on deducibility properties, [7] 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 [7] sets up the framework 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 protocol they considered.

One of the main contributions of this work is to axiomatize the if_then_else_ constructor, as the axioms introduced in [7] are not sufficient to derive equivalence of branching terms in general. We illustrate through a number of perhaps surprising examples in Section 7 to indicate the power of the axioms. They are basic, general axioms, *not designed* for with any particular protocol on our minds. We present a restricted completeness theorem for the axiomatization of if _ then _ else _ . The set of axioms is not complete in general, but we do believe that they cover most situations relevant for protocol equivalence in general. The axioms are independent. They are also modular and expansion of the logic 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.

In the BC framework, the DDH assumption appears as an axiom and it is a direct translation of the usual computational DDH assumption to our syntax, hence computational soundness of it is almost trivial. A feature of our axiomatic approach to the DDH assumption is worth noting. Recall that in the computational model, the DDH assumption for two parties is sufficient to derive the DDH assumption for multiple parties. We show that this proof can be carried out completely symbolically in our formalism (See Example 19).

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 2 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)* [20] of digital signatures, and show that the technique and our axioms can also be used to verify a trace property: *authentication*. Towards this end, we present an authenticated version of the DH protocol, which is a simplified version of the station-to-station protocol, 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.

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 [7], the two are equivalent. The IND-CCA1 and IND-CCA2 axioms are new. We also illustrate with an example how to use IND-CCA2 axiom, and finally state real-or-random secrecy and authentication verification results concerning the Needham-Schroeder-Lowe (NSL) protocol [**?**].

We would like to highlight that in the BC framework based on indistinguishability, the standard cryptographic notions seem to translate very easily 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 [7] designed this technique for indistinguishability properties, it can also be conveniently used for trace properties such as authentication.

In summary, the contributions of this work are many of the Core Axioms of Table 1, the independence theorem, the completeness theorem, the DDH axiom, the UF-CMA axiom, the IND-CCA1 and IND-CCA2 axioms, and techniques of how to apply these axioms. The role of the examples of Sections 7 and 12 as well as the DH protocol proofs and the NSL result are to demonstrate the power of the axioms we introduced, to show various techniques of applying them, and to show how to apply the technique for multiple sessions, and how to formulate in the BC framework for indistinguishability, with the help of oracles, the notions of real-or-random secrecy and even of trace properties as authentication.

Related Work. There are other attempts in the literature for computationally sound analysis of Diffie-Hellman-based protocols. Most notably, in [19], 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 has also been used to verify signed Diffie-Hellman key exchange protocols. AKE protocols have been verified using the EasyCrypt proof assistant, with Computational DH assumption [9]. The most notable other computationally sound verification of the NSL protocol is [23], but the author could not treat the case when agents play both initiator and responder roles in matching sessions, which does not cause any problem in the BC technique.

Acknowledgements. We are indebted to Hubert Comon-Lundh and Adrien Koutsos for the invaluable discussions. We also thank anonymous reviewers who have provided useful comments.

2. SYNTAX

We shall follow closely the notation in [7]. We summarize the salient features of the syntax and the semantics of the logic below, and the reader is referred to [7] 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 $\mathsf{type}(f) = (s_1, \ldots, s_n, s)$, we also write $f : s_1 \times \ldots \times s_n \to s$ and call n the *arity* of f. We assume that $\mathcal{F}$ includes at least the following function symbols:

- Booleans $\quad$ ***true*** : bool $\quad$ ***false*** : bool.
- Polymorphic conditional branching if _ then _ else _ :
  bool × message × message → message
  bool × bool × bool → bool.
- Polymorphic equality test
  $\mathsf{EQ}(\_,\_)$: message × message → bool
  bool × bool → bool.

We also use the following abbreviations

- $\mathsf{not}(b) \stackrel{\text{def}}{\equiv}$ if b then ***false*** else ***true*** .
- $x = y \stackrel{\text{def}}{\equiv} \mathsf{EQ}(x, y) \sim$ ***true***.
  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 Section 5).

EXAMPLE 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:

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

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 $g^a \stackrel{\text{def}}{\equiv} \mathtt{exp}_G(g, a)$ 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
  $\mathtt{ggen}(\_)$: message → message
- generate exponent (the "r" stands for ring)
  $\mathtt{r}(\_)$: message → 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 $\mathtt{G}(\_)$ for $\pi_1(\mathtt{ggen}(\_))$, and $\mathtt{g}(\_)$ for $\pi_2(\mathtt{ggen}(\_))$. 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, ... to abbreviate a term of the form $\mathtt{g}(x)$. We shall also use the variables a, b, c, d, ... to abbreviate terms of the form $\mathtt{r}(x)$ in the exponents of g's. When $\mathtt{ggen}(\_)$ and $\mathtt{r}(\_)$ 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: $\quad \mathtt{new}$: message
- specify action: $\quad \mathtt{act}(\_)$: message → message
- message body: $\quad \mathtt{m}(\_)$: message → message.

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

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

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

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 $\forall x.\theta[\mathtt{g}(x)]$, $\forall a.\theta[a]$ is an abbreviation for $\forall x.\theta[\mathtt{r}(x))]$ (similarly for b). □

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 $f(t_1, ..., t_n)$, where $f \in \mathcal{G}$ and $t_1, ..., t_n$ 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 $\vec{f} : \mathsf{message}^n \to \mathsf{message}^m$ to denote a vector of functions $\{f_i : \mathsf{message}^n \to \mathsf{message}\}_{i=1}^m$.

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 $n \in T_{\mathsf{message}}(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$, and if $x \in \mathcal{X}$ has sort s, then $x \in T_s(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$ and
- if $f : s_1 \times \ldots \times s_n \to s$ 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. Given $\mathcal{F}$ as defined in Example 1, variables g, a, and $f \in \mathcal{G}$, then

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

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. In order to display the formulas more concisely, we use the abbreviations

if b then $t \stackrel{\text{def}}{\equiv}$ if b then t else $\mathbf{0}$
b_1 & $b_2 \stackrel{\text{def}}{\equiv}$ if b_1 then b_2 else ***false*** □

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 $(s_1 \times \ldots \times s_n)^2$, 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 the *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*.

EXAMPLE 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. □

3. SEMANTICS

In the BC technique, two semantics are considered for the first-order formulas. The first is computational semantics: in order 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 [7].

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

- 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 $g \in \mathcal{G}$, $[\![g]\!](d_1, \ldots, d_n)$ 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

    $$[\![\mathsf{if\ \_\ then\ \_\ else\ \_}]\!] : D_{\mathrm{b}} \times D_{\mathsf{m}} \times D_{\mathsf{m}} \to D_{\mathsf{m}}$$

    such that on the triple $(d, d_1, d_2) \in D_{\mathrm{b}} \times D_{\mathsf{m}} \times D_{\mathsf{m}}$, it gives the algorithm $[\![\mathsf{if\ \_\ then\ \_\ else\ \_}]\!](d, d_1, d_2)$ with

    $$[\![\mathsf{if\ \_\ then\ \_\ else\ \_}]\!](d, d_1, d_2)(w; \rho_1; \rho_2)$$
    $$:= \begin{cases} d_1(w; \rho_1; \rho_2) & \text{if } \ d(w; \rho_1; \rho_2) = 1 \\ d_2(w; \rho_1; \rho_2) & \text{if } \ d(w; \rho_1; \rho_2) = 0 \end{cases}$$

    If $d_1, d_2 \in D_{\mathrm{b}}$, then $[\![\mathsf{if\_then\_else\_}]\!](d, d_1, d_2) \in D_{\mathrm{b}}$.
  - $\mathsf{EQ}(\_, \_)$ is interpreted as the function

    $$[\![\mathsf{EQ}(\_, \_)]\!] : D_{\mathsf{m}} \times D_{\mathsf{m}} \to D_{\mathrm{b}}$$

    such that $[\![\mathsf{EQ}(\_, \_)]\!](d_1, d_2)$ is the algorithm

    $$[\![\mathsf{EQ}(\_, \_)]\!](d_1, d_2)(w; \rho_1; \rho_2)$$
    $$:= \begin{cases} 1 & \text{if } \ d_1(w; \rho_1; \rho_2) = d_2(w; \rho_1; \rho_2) \\ 0 & \text{if } \ d_1(w; \rho_1; \rho_2) \neq d_2(w; \rho_1; \rho_2) \end{cases}$$

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.

EXAMPLE 4. We have introduced a number of function symbols in Example 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 assume that whatever the interpretations are, they operate on bit strings, and they satisfy the equations we assumed about them. It is notable that for the DDH assumption we need randomly generated groups (group schemes, see [14]), 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 [7] 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 [7] 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 the general soundness result (Theorem 1 of [7]) 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 [7]. 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

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

  - $\vec{x} \equiv x_1, \ldots, x_n$ and x are variables.
  - $N_0, N_1, ..., N_n, N$ are lists of names.
  - θ 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 $q, (N_0, N_1, ..., N_n), (\vec{x}) \xrightarrow{\theta} q', (N_0, N_1, ..., N_n, N), s, (\vec{x}, x)$ is a guarded transition which 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 put 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 5. The Diffie-Hellman key exchange protocol is the following (see e.g. [14]):

- 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 [7] 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 ($\mathsf{message}$ constants in $\mathcal{F}$): two for agent A: i_1, i_2, and two for agent B: i_3 and i_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 $\mathtt{to}$ 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 $\mathtt{to}$ can give an error. We also assume that the session identifiers are distinct: $\mathsf{EQ}(i, j) \sim \textbf{\textit{false}}$ for sessions $i \not\equiv j$. 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 i is the following:

- A receives a message into x_1
- If $\mathtt{to}(x_1) = i$, 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 $\mathtt{to}(x_2) = i$, then A computes $\mathtt{m}(x_2)^a$ and sends $\mathtt{acc}$.

The responder role of A is the following:

- A receives a message into y_1
- If $\mathtt{to}(y_1) = i$, 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_j numbers the rounds of session i_j that has been completed if it is an initiator session, and ℓ_j numbers the round of session i_j that has been completed if it is a responder session. Clearly, both k_j and l_j cannot be 0 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 j, either k_j or l_j is maximal are also final.

The transition system is then such that there is a transition corresponding to each pair $(q^{k_1k_2k_3k_4}_{\ell_1\ell_2\ell_3\ell_4}, q^{k'_1k'_2k'_3k'_4}_{\ell'_1\ell'_2\ell'_3\ell'_4})$ 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 test fails, 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:

$$q^{1002}_{0100},(\vec{N}),(\vec{x}) \xrightarrow{\mathsf{EQ}(\mathtt{to}(x_5),i_1)} q^{2002}_{0100},(\vec{N},()),\mathtt{acc},(\vec{x},x_5)$$

$$q^{1002}_{0100},(\vec{N}),(\vec{x}) \xrightarrow{\mathsf{EQ}(\mathtt{to}(x_5),i_3)\&\mathsf{EQ}(\mathtt{act}(x_5),\mathtt{new})} \to q^{1012}_{0100},(\vec{N},(n_5)),\mathtt{g}(n_0)^{\mathtt{r}(n_5)},(\vec{x},x_5)$$

$$q^{1002}_{0100},(\vec{N}),(\vec{x}) \xrightarrow{\mathsf{EQ}(\mathtt{to}(x_5),i_3)} q^{1002}_{0110},(\vec{N},(n_5)),\mathtt{g}(n_0)^{\mathtt{r}(n_5)},(\vec{x},x_5)$$

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

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_j 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 $j = 1,2,3,4$

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

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

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

That is, if the adversary calls for a new session i_j, then a new initiator session is started. If the adversary sends to sessions i_j but does not call for a new session, then the agent starts a responder session, and assumes the incoming message is from the initiator.

4.2 Execution and indistinguishability

Computational and symbolic executions were defined precisely in [7]. 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. [16]).However, as the authors discussed in [7], obtaining computational soundness through such matching seems infeasible. Instead, the authors in [7] *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 6. The folded symbolic execution of two sessions of the DH protocol between A initiator and B responder has the following trace:

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

if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_1)$ & $\mathsf{EQ}(\mathtt{act}(f_1(\phi_1)),\mathtt{new})$
then g^{a_1}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_1)$
then g^{a_1}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_2)$ & $\mathsf{EQ}(\mathtt{act}(f_1(\phi_1)),\mathtt{new})$
then g^{a_2}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_2)$
then g^{a_2}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_3)$ & ...
⋮
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_4)$
then g^{a_4}
else $\mathbf{0}$

- etc.

Where $G \equiv \pi_1(\mathtt{ggen}(n_0))$ and $g \equiv \pi_2(\mathtt{ggen}(n_0))$ and $a_j \equiv \mathtt{r}(n_{m(j)})$ for $j = 1,2,3,4$ and some increasing function $m : \mathbb{N} \to \mathbb{N}$. 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:

if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_1)$ & $\mathsf{EQ}(\mathtt{act}(f_1(\phi_1)),\mathtt{new})$
then q^{1000}_{0000}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_1)$
then q^{0000}_{1000}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_2)$ & $\mathsf{EQ}(\mathtt{act}(f_1(\phi_1)),\mathtt{new})$
then q^{0100}_{0000}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_2)$
then q^{0000}_{0100}
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_3)$ & ...
⋮
else if $\mathsf{EQ}(\mathtt{to}(f_1(\phi_1)),i_4)$
then q^{0000}_{0001}
else $\mathbf{0}$

Then, the states have to be replaced with the terms that describe the transitions out of the state. For example, in Example 5, we also listed the transitions from q^{1002}_{0100}. The term that corresponds to these transitions from q^{1002}_{0100} is

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, $\vec{x} \sim \vec{y} \longrightarrow p(\vec{x}) \sim p(\vec{y})$
FUNCAPP:	for any $\vec{f} : s_1, ..., s_n \to s'_1, ..., s'_m, \vec{f} \in \mathcal{F} \cup \mathcal{G}$, $\vec{x} \sim \vec{y} \longrightarrow \vec{x}, \vec{f}(\vec{x}) \sim \vec{y}, \vec{f}(\vec{y})$
TFDIST:	$\neg$ (***true*** $\sim$ ***false***)

Axioms for equality.

EQREFL:	$x = x$
EQCONG:	$=$ is a congruence relation with respect to the current syntax.
EQTHEO:	$=$ preserves the equational theory of functions presuming that the computational interpretation satisfies the equations bitwise.

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 $t_1[$***true***$]$ else $t_2[$***false***$]$
IFTRUE:	if ***true*** then x else $y \;=\; x$
IFFALSE:	if ***false*** then x else $y \;=\; y$
IFBRANCH:	$\vec{z}, b, x \sim \vec{z}', b', x' \;\wedge\; \vec{z}, b, y \sim \vec{z}', b', y' \longrightarrow$ $\vec{z}, b,$ if b then x else y $\sim \vec{z}', b',$ if b' then x' else y'

Axioms for names.

FRESHIND:	for any names n_1, n_2 and lists of closed terms $\vec{v}, \vec{w}$, such that fresh$(n_1; \vec{v}, \vec{w})$ and 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 fresh$(n; v)$ holds, we have EQ$(n, v) \sim$ ***false***.

Table 1: Core Axioms

if EQ$(\mathtt{to}(f_4(\phi_4)), i_1)$
then acc
else if EQ$(\mathtt{to}(f_4(\phi_4)), i_3)$ & EQ$(\mathtt{act}(f_4(\phi_4)), \mathtt{new})$
then g^{b_1}
else if EQ$(\mathtt{to}(f_4(\phi_4)), i_4)$then g^{b_1} else $\mathbf{0}$. □

This way, the indistinguishability of two protocols Π and Π' can be reduced to the indistinguishability of the lists of the sent messages. Let fold(Π) denote the folded execution of protocol Π, and let $\Phi($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 [7]:

THEOREM 1. *Let Π, Π' be two protocols. Let $\mathfrak{A}$ be any set of formulas (axioms). If $\mathfrak{A}$ and $\Phi($**fold**$(\Pi)) \not\sim \Phi($**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 [7] 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 tare 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 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* [7]: For a list of pairwise distinct names N, and a (possibly empty) list of closed terms $\vec{v}$, 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 [7]. 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:

THEOREM 2. *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 complete. A complete axiomatization might be very difficult, and it is not a priority. Completeness in restricted cases can be shown though. For example, the following theorem is true:

THEOREM 3. *Suppose the only function symbols in t_1 and t_2 are* if _ then _ else _ *,* ***true*** *and* ***false****.*

$t_1 = t_2$ is computationally valid, if and only if
EQREFL, EQCONG,
IFSAME, IFEVALIFTRUE, 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

$$t_1 = \mathsf{if}\ b\ \mathsf{then}\ t_1^1\ \mathsf{else}\ t_1^2$$

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

$$t_2 = \mathsf{if}\ b\ \mathsf{then}\ t_2^1\ \mathsf{else}\ t_2^2$$

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 $t_2 = \mathsf{if}\ b\ \mathsf{then}\ t_2^1\ \mathsf{else}\ t_2^2$ with $t_1^1 = t_2^1 = t_2$. 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

$$\tfrac{1}{2}\mathbf{Prob}\{\rho : [\![t_1^1]\!]^\sigma_{\eta,\rho} \neq [\![t_2^1]\!]^\sigma_{\eta,\rho}\} \leq \mathbf{Prob}\{\rho : [\![t_1]\!]^\sigma_{\eta,\rho} \neq [\![t_2]\!]^\sigma_{\eta,\rho}\},$$

and since the LHS of this inequality is non-negligible, the RHS 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

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

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 7. For the function symbols in Example 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.

EXAMPLE 8. Let us define the following three axioms

$$\text{IfIdemp}: \quad \mathsf{if}\ b\ \mathsf{then}\ \begin{pmatrix}\mathsf{if}\ b\ \mathsf{then}\ x_1\\ \mathsf{else}\ y_1\end{pmatrix}\ \mathsf{else}\ \begin{pmatrix}\mathsf{if}\ b\ \mathsf{then}\ x_2\\ \mathsf{else}\ y_2\end{pmatrix} = \mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ y_2$$

$$\text{IfMorph}: \quad f(z_1, ..., \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y\ , ..., z_n) = \begin{array}{r}\mathsf{if}\ b\ \mathsf{then}\ f(z_1, ..., x, ..., z_n)\\ \mathsf{else}\ f(z_1, ..., y, ..., z_n)\end{array}$$

$$\text{IfTF}: \quad \mathsf{if}\ b\ \mathsf{then}\ \textbf{\textit{true}}\ \mathsf{else}\ \textbf{\textit{false}} = b$$

It is easy to see that

$$\text{IfTF}, \text{IfIdemp}, \text{IfMorph}, \text{EqRefl}, \text{EqCong} \vdash \text{IfSame}, \text{IfEval}$$

and

$$\text{IfEval}, \text{IfSame}, \text{IfTrue}, \text{IfFalse}, \text{EqRefl}, \text{EqCong} \vdash \text{IfTF}, \text{IfIdemp}, \text{IfMorph}\ .$$

To see the first, we need the transitivity of equality with:

$$\begin{array}{rl} & \mathsf{if}\ b\ \mathsf{then}\ t_1[b]\ \mathsf{else}\ t_2[b]\\ \overset{\text{IfTF}}{=} & \begin{array}{l}\mathsf{if}\ b\ \mathsf{then}\ t_1[\mathsf{if}\ b\ \mathsf{then}\ \textbf{\textit{true}}\ \mathsf{else}\ \textbf{\textit{false}}\ ]\\ \quad \mathsf{else}\ t_2[\mathsf{if}\ b\ \mathsf{then}\ \textbf{\textit{true}}\ \mathsf{else}\ \textbf{\textit{false}}\ ]\end{array}\\ \overset{\text{IfMorph}}{=} & \begin{array}{l}\mathsf{if}\ b\ \mathsf{then}\ \mathsf{if}\ b\ \mathsf{then}\ t_1[\textbf{\textit{true}}]\ \mathsf{else}\ t_1[\textbf{\textit{false}}\ ]\\ \quad \mathsf{else}\ \mathsf{if}\ b\ \mathsf{then}\ t_2[\textbf{\textit{true}}]\ \mathsf{else}\ t_2[\textbf{\textit{false}}\ ]\end{array}\\ \overset{\text{IfIdemp}}{=} & \mathsf{if}\ b\ \mathsf{then}\ t_1[\textbf{\textit{true}}]\ \mathsf{else}\ t_2[\textbf{\textit{false}}] \end{array}$$

and[1]

$$\begin{array}{rl} x & \overset{\text{IfTrue}}{=} \mathsf{if}\ \textbf{\textit{true}}\ \mathsf{then}\ x\ \mathsf{else}\ \mathsf{if}\ b\ \mathsf{then}\ y\ \mathsf{else}\ z\\ & \overset{\text{IfMorph}}{=} \begin{array}{l}\mathsf{if}\ b\ \mathsf{then}\ \mathsf{if}\ \textbf{\textit{true}}\ \mathsf{then}\ x\ \mathsf{else}\ y\\ \quad \mathsf{else}\ \mathsf{if}\ \textbf{\textit{true}}\ \mathsf{then}\ x\ \mathsf{else}\ z\end{array}\\ & \overset{\text{IfTrue}}{=} \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ x \end{array}$$

To see the second:

$$b \overset{\text{IfSame}}{=} \mathsf{if}\ b\ \mathsf{then}\ b\ \mathsf{else}\ b \overset{\text{IfEval}}{=} \mathsf{if}\ b\ \mathsf{then}\ \textbf{\textit{true}}\ \mathsf{else}\ \textbf{\textit{false}}$$

$$\begin{array}{rl} & f(z_1, ..., \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y\ , ..., z_n)\\ \overset{\text{IfSame}}{=} & \begin{array}{l}\mathsf{if}\ b\ \mathsf{then}\ f(z_1, ..., \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y\ , ..., z_n)\\ \quad \mathsf{else}\ f(z_1, ..., \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y\ , ..., z_n)\end{array}\\ \overset{\text{IfEval}}{=} & \begin{array}{l}\mathsf{if}\ b\ \mathsf{then}\ f(z_1, ..., \mathsf{if}\ \textbf{\textit{true}}\ \mathsf{then}\ x\ \mathsf{else}\ y\ , ..., z_n)\\ \quad \mathsf{else}\ f(z_1, ..., \mathsf{if}\ \textbf{\textit{false}}\ \mathsf{then}\ x\ \mathsf{else}\ y\ , ..., z_n)\end{array}\\ \overset{\substack{\text{IfTrue}\\ \text{IfFalse}}}{=} & \mathsf{if}\ b\ \mathsf{then}\ f(z_1, ..., x, ..., z_n)\ \mathsf{else}\ f(z_1, ..., y, ..., z_n) \end{array}$$

$$\begin{array}{rl} & \mathsf{if}\ b\ \mathsf{then}\ \begin{pmatrix}\mathsf{if}\ b\ \mathsf{then}\ x_1\\ \mathsf{else}\ y_1\end{pmatrix}\ \mathsf{else}\ \begin{pmatrix}\mathsf{if}\ b\ \mathsf{then}\ x_2\\ \mathsf{else}\ y_2\end{pmatrix}\\ \overset{\text{IfEval}}{=} & \mathsf{if}\ b\ \mathsf{then}\ \begin{pmatrix}\mathsf{if}\ \textbf{\textit{true}}\ \mathsf{then}\ x_1\\ \mathsf{else}\ y_1\end{pmatrix}\ \mathsf{else}\ \begin{pmatrix}\mathsf{if}\ \textbf{\textit{false}}\ \mathsf{then}\ x_2\\ \mathsf{else}\ y_2\end{pmatrix}\\ \overset{\substack{\text{IfTrue}\\ \text{IfFalse}}}{=} & \mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ y_2\ . \end{array}$$ □

[1] This observation is due to Adrien Koutsos

5.1 Soundness of the Axioms

The soundness of the axioms for indistinguishability were proven in [7] 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 1. *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***, $\mathcal{M}^c \models \mathsf{EQ}(x, y) \sim$ ***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 or 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 2. *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 $[\![\mathsf{if\ \_\ then\ \_\ else\ \_}]\!]$, it is a trivial matter to verify that

$$[\![t_1]\!]^\sigma_{\eta,\rho} = [\![t_2]\!]^\sigma_{\eta,\rho}. \ .$$

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

IFBRANCH:

$$\vec{z}, b, x \sim \vec{z}', b', x' \ \wedge \ \vec{z}, b, y \sim \vec{z}', b', y' \longrightarrow \\ \vec{z}, b, \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y \ \sim \ \vec{z}', b', \mathsf{if}\ b'\ \mathsf{then}\ x'\ \mathsf{else}\ y'$$

Assume an assignment σ of the free variables $\vec{z}, \vec{z}', b, b', x, x', y, y'$, taking values in the corresponding domains and a security parameter η. Assume further that $\vec{z}, b, x \sim \vec{z}', b', x'$ and $\vec{z}, b, y \sim \vec{z}', b', y'$. Fix an adversary $\mathcal{A}$. Let $p_l, p_r, p_{x_1}, p_{x_2}, p_{y_1}$ and p_{y_2} be defined as follows.

$$p_l = \mathbf{Prob}\{\rho\colon \mathcal{A}([\![\vec{z}, b, \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y\ ]\!]^\sigma_{\rho,\eta}; \rho_2) = 1\}.$$
$$p_r = \mathbf{Prob}\{\rho\colon \mathcal{A}([\![\vec{z}', b', \mathsf{if}\ b'\ \mathsf{then}\ x'\ \mathsf{else}\ y'\ ]\!]^\sigma_{\rho,\eta}; \rho_2) = 1\}.$$
$$p_x = \mathbf{Prob}\{\rho : \mathcal{A}([\![\vec{z}, b, x]\!]^\sigma_{\rho,\eta}; \rho_2) = 1 \ \&\ [\![b]\!]^\sigma_{\rho,\eta} = 1\}$$
$$p_{x'} = \mathbf{Prob}\{\rho : \mathcal{A}([\![\vec{z}', b', x']\!]^\sigma_{\rho,\eta}; \rho_2) = 1 \ \&\ [\![b']\!]^\sigma_{\rho,\eta} = 1\}$$
$$p_y = \mathbf{Prob}\{\rho : \mathcal{A}([\![\vec{z}, b, y]\!]^\sigma_{\rho,\eta}; \rho_2) = 1 \ \&\ [\![b]\!]^\sigma_{\rho,\eta} = 0\}.$$
$$p_{y'} = \mathbf{Prob}\{\rho : \mathcal{A}([\![\vec{z}', b', y']\!]^\sigma_{\rho,\eta}; \rho_2) = 1 \ \&\ [\![b']\!]^\sigma_{\rho,\eta} = 0\}.$$

It is easy to see that $p_l = p_x + p_y$ and that $p_r = p_{x'} + p_{y'}$. Therefore $|p_l - p_r| \leq |p_x - p_{x'}| + |p_y - p_{y'}|$. In order to prove the soundness of the axiom we need to show that $|p_l - p_r|$ is negligible in η. In order to prove this, it suffices to show that both $|p_x - p_{x'}|$ are $|p_{y_1} - p_{y'}|$ 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 $\vec{m}^1, b_1, m_2$ and random tape ρ_2 runs $\mathcal{A}(\vec{m}^1, b_1, m_2)$ when b_1 is 1 and outputs 0 otherwise. Now, it is easy to see that

$$\mathbf{Prob}\{\rho : \mathcal{B}([\![\vec{z}, b, x]\!]^\sigma_{\rho,\eta}; \rho_2) = 1\} \quad = \quad p_x$$
$$\mathbf{Prob}\{\rho : \mathcal{B}([\![\vec{z}', b', x']\!]^\sigma_{\rho,\eta}; \rho_2) = 1\} \quad = \quad p_{x'}.$$

Thus, $|p_x - p_{x'}| = |\mathbf{Prob}\{\rho : \mathcal{B}([\![\vec{z}, b, x]\!]^\sigma_{\rho,\eta}; \rho_2) = 1\} - \mathbf{Prob}\{\rho : \mathcal{B}([\![\vec{z}', b', x']\!]^\sigma_{\rho,\eta}; \rho_2) = 1\}|$. Now, the latter is negligible in η as $\vec{z}, b, x \sim \vec{z}', b', x'$. 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 3. *Axioms* FRESHIND *and* FRESHNEQ *are computationally sound.*

PROOF. For FRESHIND, note that $\mathsf{fresh}(n_1, n_2; \vec{v}, \vec{w})$ 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 $[\![\mathsf{EQ}(n, v)]\!]$ 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 4. *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 [14]).*

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 [13], Definition 2.1. □

7. SHORT EXAMPLES

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

EXAMPLE 9. In the formula below, IFMORPH lets us pull out $\mathsf{if\ \_\ then\ \_\ else\ \_}$ 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

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

EXAMPLE 10. We have that for any constant $f \in \mathcal{F} \cup \mathcal{G}$,

$$\begin{array}{l}\textsc{Trans}, \textsc{Restr}, \textsc{FuncApp}, \textsc{EqRefl} \vdash \\ x \sim f \longrightarrow x = f.\end{array}$$

To see this, consider $x \sim f$. By FUNCAPP, $x, f \sim f, f$, and again by FUNCAPP, $x, f, \mathsf{EQ}(x, f) \sim f, f, \mathsf{EQ}(f, f)$. By RESTR, $\mathsf{EQ}(x, f) \sim \mathsf{EQ}(f, f)$. By EQREFL, $f = f$, which is a shorthand for $\mathsf{EQ}(f, f) \sim \boldsymbol{true}$. Then by TRANS, $\mathsf{EQ}(x, f) \sim \boldsymbol{true}$, which is $x = f$.

Note that in particular, $x = y$ iff $\mathsf{EQ}(x, y) = \boldsymbol{true}$. □

EXAMPLE 11. We have

$$\begin{array}{l}\textsc{Trans}, \textsc{Restr}, \textsc{FuncApp}, \\ \textsc{IfSame}, \textsc{IfIdemp}, \textsc{IfMorph}, \textsc{IfTF}, \textsc{EqRefl}, \textsc{EqCong} \vdash \\ \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ x_1\ \mathsf{else}\ y\ = \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ x_2\ \mathsf{else}\ y\end{array}$$

This is because:

$$\begin{array}{ll} & \mathsf{EQ}\left(\begin{array}{l}\mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ x_1\ \mathsf{else}\ y\ , \\ \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ x_2\ \mathsf{else}\ y\end{array}\right) \\ \overset{\textsc{IfMorph}}{=} & \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ \mathsf{EQ}\left(\begin{array}{l}x_1, \\ \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ x_2\ \mathsf{else}\ y\end{array}\right) \\ & \mathsf{else}\ \mathsf{EQ}\left(\begin{array}{l}y, \\ \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ x_2\ \mathsf{else}\ y\end{array}\right) \\ \overset{\text{Example 9}}{=} & \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ \mathsf{EQ}\ (x_1, x_2)\ \ \mathsf{else}\ \mathsf{EQ}\ (y, y) \\ \overset{\text{Example 10}}{=} & \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ \mathsf{EQ}\ (x_1, x_2)\ \ \mathsf{else}\ \boldsymbol{true} \\ \overset{\textsc{IfEval}}{=} & \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{true}\ \overset{\textsc{IfSame}}{=}\ \boldsymbol{true}\end{array}$$

where EQCONG is also used, but we omitted its indication. □

EXAMPLE 12. We prove the following:

$$\begin{array}{l}\textsc{IfIdemp}, \textsc{IfMorph}, \textsc{EqRefl}, \textsc{EqCong} \vdash \\ \mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ y_1\ = \mathsf{if}\ b\ \mathsf{then}\ x_2\ \mathsf{else}\ y_2 \\ \quad \longrightarrow \mathsf{if}\ b\ \mathsf{then}\ t[x_1]\ \mathsf{else}\ t'[y_1]\ = \mathsf{if}\ b\ \mathsf{then}\ t[x_2]\ \mathsf{else}\ t'[y_2]\end{array}$$

The statement can be proven using Example 9 and congruence of the equality:

$$\begin{array}{l}\mathsf{if}\ b\ \mathsf{then}\ t[x_1]\ \mathsf{else}\ t'[y_1] \\ = \mathsf{if}\ b\ \mathsf{then}\ t[\mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ y_1\ ]\ \mathsf{else}\ t'[\mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ y_1\ ] \\ = \mathsf{if}\ b\ \mathsf{then}\ t[\mathsf{if}\ b\ \mathsf{then}\ x_2\ \mathsf{else}\ y_2\ ]\ \mathsf{else}\ t'[\mathsf{if}\ b\ \mathsf{then}\ x_2\ \mathsf{else}\ y_2\ ] \\ = \mathsf{if}\ b\ \mathsf{then}\ t[x_2]\ \mathsf{else}\ t'[y_2]\end{array}$$

Putting this together with Example 11, we have in particular that:[2]

$$\begin{array}{l}\textsc{EqBranch} \equiv \\ \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ t[x_1]\ \mathsf{else}\ t' = \mathsf{if}\ \mathsf{EQ}(x_1, x_2)\ \mathsf{then}\ t[x_2]\ \mathsf{else}\ t'.\end{array}$$

□

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.

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

EXAMPLE 13. By the previous examples, we have $\mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false}) = \boldsymbol{false}$ as follows:

$$\begin{array}{ll} & \mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false}) \\ \overset{\textsc{IfTF}}{=} & \mathsf{if}\ \mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false})\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{EqBranch}}{=} & \mathsf{if}\ \mathsf{EQ}(\boldsymbol{true}, \boldsymbol{false})\ \mathsf{then}\ \boldsymbol{false}\ \mathsf{else}\ \boldsymbol{false}\ \overset{\textsc{IfSame}}{=} \boldsymbol{false}\end{array}$$

□

EXAMPLE 14. By the previous examples, we also have

$$\mathsf{EQ}(x, y) = \mathsf{EQ}(y, x).$$

The proof is the following:

$$\begin{array}{ll} & \mathsf{EQ}(x, y) \\ \overset{\textsc{IfTF}}{=} & \mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{IfEval}}{=} & \mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ \mathsf{EQ}(x, y)\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{EqBranch}}{=} & \mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ \mathsf{EQ}(y, x)\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{IfTF}}{=} & \mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ (\mathsf{if}\ \mathsf{EQ}(y, x)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false}\ ) \\ & \qquad \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{IfMorph}}{=} & \mathsf{if}\ \mathsf{EQ}(y, x)\ \mathsf{then}\ (\mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false}\ ) \\ & \qquad \mathsf{else}\ (\mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ \boldsymbol{false}\ \mathsf{else}\ \boldsymbol{false}\ ) \\ \overset{\textsc{IfSame}}{=} & \mathsf{if}\ \mathsf{EQ}(y, x)\ \mathsf{then}\ (\mathsf{if}\ \mathsf{EQ}(x, y)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false}\ ) \\ & \qquad \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{IfTF}}{=} & \mathsf{if}\ \mathsf{EQ}(y, x)\ \mathsf{then}\ \mathsf{EQ}(x, y)\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{EqBranch}}{=} & \mathsf{if}\ \mathsf{EQ}(y, x)\ \mathsf{then}\ \mathsf{EQ}(y, x)\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{IfEval}}{=} & \mathsf{if}\ \mathsf{EQ}(y, x)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false}\ \overset{\textsc{IfTF}}{=}\ \mathsf{EQ}(y, x)\end{array}$$ □

EXAMPLE 15. 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 10, means $\mathsf{EQ}(n_2, \pi_2(\langle n_1, n_2\rangle)) = \boldsymbol{true}$. Then

$$\begin{array}{ll} & \mathsf{EQ}(n_1, \langle n_1, n_2\rangle) \\ \overset{\textsc{IfTF}}{=} & \mathsf{if}\ \mathsf{EQ}(n_1, \langle n_1, n_2\rangle)\ \mathsf{then}\ \boldsymbol{true}\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\substack{\textsc{EqTheo} \\ \text{Example 10}}}{=} & \mathsf{if}\ \mathsf{EQ}(n_1, \langle n_1, n_2\rangle)\ \mathsf{then}\ \mathsf{EQ}(n_2, \pi_2(\langle n_1, n_2\rangle))\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\textsc{EqBranch}}{=} & \mathsf{if}\ \mathsf{EQ}(n_1, \langle n_1, n_2\rangle)\ \mathsf{then}\ \mathsf{EQ}(n_2, \pi_2(n_1))\ \mathsf{else}\ \boldsymbol{false} \\ \overset{\substack{\textsc{FreshNEq} \\ \text{Example 10}}}{=} & \mathsf{if}\ \mathsf{EQ}(n_1, \langle n_1, n_2\rangle)\ \mathsf{then}\ \boldsymbol{false}\ \mathsf{else}\ \boldsymbol{false}\ \overset{\textsc{IfSame}}{=} \boldsymbol{false}\end{array}$$ □

EXAMPLE 16. It is easy to see from the definition of not, that

$$\begin{array}{l}\textsc{EqRefl}, \textsc{EqCong}, \textsc{IfMorph}, \textsc{IfTrue}, \textsc{IfFalse} \vdash \\ \mathsf{if}\ \mathsf{not}(b)\ \mathsf{then}\ x\ \mathsf{else}\ y\ = \mathsf{if}\ b\ \mathsf{then}\ y\ \mathsf{else}\ x\end{array}$$ □

EXAMPLE 17. For variables $b_1, b_2, n_1, n_1', n_2, n_2', n_2, n_3', n_4'$:

$$\begin{array}{l}b_1, b_2, n_1, n_2, n_3 \sim b_1, b_2, n_1', n_2', n_3' \sim b_1, b_2, n_4', n_2', n_3' \rightarrow \\ \mathsf{if}\ b_1\ \mathsf{then}\ n_1\ \mathsf{else}\ (\mathsf{if}\ b_2\mathsf{then}\ \langle n_2, n_3\rangle\mathsf{else}\ \langle n_1, n_2, n_3\rangle)\ \sim \\ \mathsf{if}\ \mathsf{not}(b_2)\ \mathsf{then}\ (\mathsf{if}\ b_1\mathsf{then}\ n_1'\mathsf{else}\ \langle n_1', n_2', n_3'\rangle) \\ \qquad \mathsf{else}\ (\mathsf{if}\ b_1'\mathsf{then}\ n_4'\mathsf{else}\ \langle n_2', n_3'\rangle)\end{array}$$

Fix $b_1, b_1', b_2, b_2', n_1, n_1', n_2, n_2', n_3, n_3', n_4'$.

1. Assume
   $b_1, b_2, n_1, n_2, n_3 \sim b_1, b_2, n_1', n_2', n_3' \sim b_1, b_2, n_4', n_2', n_3'$.
2. Thanks to IfMorph, we have that
   $\mathsf{if}\ b_1 \mathsf{then}\ n_1\ \mathsf{else}\ (\mathsf{if}\ b_2 \mathsf{then}\ \langle n_2, n_3\rangle \mathsf{else}\ \langle n_1, n_2, n_3\rangle)$
   $=$
   $\mathsf{if}\ b_2 \mathsf{then}\ (\mathsf{if}\ b_1\ \mathsf{then}\ n_1 \mathsf{else}\ \langle n_2, n_3\rangle)$
   $\quad \mathsf{else}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_1, n_2, n_3\rangle)$
3. From Example 16 we get that
   $\mathsf{if}\ b_2\ \mathsf{then}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_2, n_3\rangle)$
   $\quad \mathsf{else}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_1, n_2, n_3\rangle)$
   $=$
   $\mathsf{if}\ \mathsf{not}(b_2)\ \mathsf{then}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_1, n_2, n_3\rangle)$
   $\quad \mathsf{else}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_2, n_3\rangle)$
4. By line 1, axioms FuncApp and Restr,
   $\mathsf{not}(b_2), \mathsf{if}\ b_1\ \mathsf{then}\ n_1\ \mathsf{else}\ \langle n_1, n_2, n_3\rangle$
   $\quad \sim \mathsf{not}(b_2), \mathsf{if}\ b_1\ \mathsf{then}\ n_1'\ \mathsf{else}\ \langle n_1', n_2', n_3'\rangle$
   and
   $\mathsf{not}(b_2), \mathsf{if}\ b_1\ \mathsf{then}\ n_1\ \mathsf{else}\ \langle n_2, n_3\rangle$
   $\quad \sim \mathsf{not}(b_2), \mathsf{if}\ b_1\ \mathsf{then}\ n_4'\ \mathsf{else}\ \langle n_2', n_3'\rangle$.
5. The previous step together with IfBranch, Restr imply
   $\mathsf{if}\ \mathsf{not}(b_2)\ \mathsf{then}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_1, n_2, n_3\rangle)$
   $\quad \mathsf{else}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1 \mathsf{else}\ \langle n_2, n_3\rangle)$
   $\sim$
   $\mathsf{if}\ \mathsf{not}(b_2)\ \mathsf{then}\ (\mathsf{if}\ b_1 \mathsf{then}\ n_1' \mathsf{else}\ \langle n_1', n_2', n_3'\rangle)$
   $\quad \mathsf{else}\ (\mathsf{if}\ b_1' \mathsf{then}\ n_4' \mathsf{else}\ \langle n_2', n_3'\rangle)$
6. Line 5 together with the congruence of equality imply
   $\mathsf{if}\ b_1\ \mathsf{then}\ n_1\ \mathsf{else}\ (\mathsf{if}\ b_2 \mathsf{then}\ \langle n_2, n_3\rangle \mathsf{else}\ \langle n_1, n_2, n_3\rangle)$
   $\sim$
   $\mathsf{if}\ b_2'\ \mathsf{then}\ (\mathsf{if}\ b_1' \mathsf{then}\ n_1' \mathsf{else}\ \langle n_1', n_2', n_3'\rangle)$
   $\quad \mathsf{else}\ (\mathsf{if}\ b_1' \mathsf{then}\ n_4' \mathsf{else}\ \langle n_2', n_3'\rangle)$.
   The theorem follows. □

Example 18. Axioms IfIdemp and IfMorph reduce terms in the frame significantly for the following reason. Consider the simple situation when the frame ϕ_2 looks like

$$\begin{array}{l}\phi_0, \mathsf{if}\ b_1\big[f_1(\phi_0)\big]\ \mathsf{then}\ t_1^1\big[f_1(\phi_0)\big]\ \mathsf{else}\ t_1^2\big[f_1(\phi_0)\big]\ ,\\ \quad \mathsf{if}\ b_1\big[f_1(\phi_0)\big]\\ \quad \mathsf{then}\ \Big(\mathsf{if}\ b_2^1\big[f_2(\phi_1)\big]\ \mathsf{then}\ t_2^{11}\big[f_2(\phi_1)\big]\ \mathsf{else}\ t_2^{12}\big[f_2(\phi_1)\big]\Big)\\ \quad \mathsf{else}\ \Big(\mathsf{if}\ b_2^2\big[f_2(\phi_1)\big]\ \mathsf{then}\ t_2^{01}\big[f_2(\phi_1)\big]\ \mathsf{else}\ t_2^{00}\big[f_2(\phi_1)\big]\Big)\end{array}$$

Inside f_2, the ϕ_1 is $\phi_0, \mathsf{if} b_1[f_1(\phi_0)] \mathsf{then} t_1^1[f_1(\phi_0)] \mathsf{else} t_1^2[f_1(\phi_0)]$ also has branching, but by axiom IfIdemp and IfMorph, that branching can be removed. So the last term in the frame equals

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

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 19. 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
   $a, b, G, g, g^d, g^c, g^{dc} g^a, g^b, g^{ca}, g^{cb}$
   $\sim a, b, G, g, g^d, g^c, g^e g^a, g^b, g^{ca}, g^{cb}$
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 20. 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 [3]. According to their definition, the adversary can request an oracle to reveal the shared key of the honest agents. The oracle either reveals 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 chose 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 on the other hand, 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 5 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 $q^{k_1k_2k_3k_4}_{\dots\ell_j=1\dots}$, we add the following transitions:

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

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

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

  We order the transitions so that they are all of higher order than those in Section 5. The order of the transitions labeled with θ_i decreases with increasing i. If $q^{k_1k_2k_3k_4}_{\dots\ell_{j_1}=1\dots} = q^{k_1k_2k_3k_4}_{\dots\ell_{j_2}=1\dots}$, then the transition corresponding to smaller j has higher order. Moreover, to a state $q^{\dots k_j=2\dots}_{\ell_1\ell_2\ell_3\ell_4}$, we add the following transitions:

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

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

  $$\theta_{ih} \equiv \mathsf{EQ}(\mathtt{reveal}(x), i_j)\ \&\ \mathsf{EQ}(\mathtt{to}(x_i), i_j)\ \&\ \mathsf{EQ}(\mathtt{to}(x_h), i_j)\ \&\mathsf{not}(\mathsf{EQ}(\mathtt{act}(x_i), \mathtt{new}))\ \&\ \mathsf{EQ}(\mathtt{act}(x_h), \mathtt{new})$$

  We order the transitions so that they are all of higher order than those in Section 5, they decrease by j, 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

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

  if $k_j < 2$ and $\ell_j < 1$ with $\theta_j \equiv \mathsf{EQ}(\mathtt{reveal}(x), i_j)$, again with higher order than those transitions in Example 5.

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 i_α, and there was another session i_β 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:

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

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

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

  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 key match, 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 5. *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 $\mathsf{EQ}(\mathtt{reveal}(x), i_j)$ is a new condition, the condition $\mathsf{EQ}(\mathtt{to}(x_i), i_j)$ already appeared earlier in the execution, so there is already a branching according to the latter. By IFMORPH and IFIDEMP, just as in Example 18, 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 $\mathsf{EQ}(\mathtt{to}(f_i(\phi_i)), i_j)$ is satisfied. Only the branching according to $\mathsf{EQ}(\mathtt{reveal}(x), i_j)$ remains in the oracle step. When i_j 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 i_j 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 $\mathsf{EQ}(\mathtt{m}(x_\delta), \mathtt{g}(n_0)^{\mathtt{r}(n_\epsilon)})$ and $\mathsf{EQ}(\mathtt{m}(x_\gamma), \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)})$ 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 i_α, while $\mathtt{m}(x_\delta)^{\mathtt{r}(n_\delta)}$ is the key computed in Responder session i_β, conditions $\mathsf{EQ}(\mathtt{m}(x_\delta), \mathtt{g}(n_0)^{\mathtt{r}(n_\epsilon)})$ and $\mathsf{EQ}(\mathtt{m}(x_\gamma), \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)})$ 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 $\Phi(\mathsf{fold}(\Pi_1)) \sim \Phi(\mathsf{fold}(\Pi_2''))$ by several applications of axiom IFSAME. Then, consider the protocol Π_2', which we receive from Π_2 by replacing the output $\mathtt{g}(n_0)^{\mathtt{r}(n)}$ by $\mathtt{g}(n_0)^{\mathtt{r}(n_\delta)\mathtt{r}(n_\epsilon)}$. Then, according to the previous paragraph, using the results of Example 11 and Example 12, we have $\Phi(\mathsf{fold}(\Pi_2'')) \sim \Phi(\mathsf{fold}(\Pi_2'))$.

The only thing left to prove is $\Phi(\mathsf{fold}(\Pi_2')) \sim \Phi(\mathsf{fold}(\Pi_2))$. This relies mainly on the DDH axiom and the IFBRANCH axiom: The only difference between $\Phi(\mathsf{fold}(\Pi_2'))$ and $\Phi(\mathsf{fold}(\Pi_2))$ is that some of the final sent messages are $\mathtt{g}(n_0)^{\mathtt{r}(n_\delta)\mathtt{r}(n_\epsilon)}$ in the first, while $\mathtt{g}(n_0)^{\mathtt{r}(n)}$ 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 look like $\mathtt{G}(\mathtt{r}(n_0))$, $\mathtt{g}(\mathtt{r}(n_0))$, $\mathtt{g}(n_0)^{\mathtt{r}(n_1)}$, ..., $g^{\mathtt{r}(n_4)}$, $\mathtt{g}(n_0)^{\mathtt{r}(n_\delta)\mathtt{r}(n_\epsilon)}$. Because of the DDH assumption, FRESHIND, FUNCAPP, we have

$$\mathtt{G}(\mathtt{r}(n_0)), \mathtt{g}(\mathtt{r}(n_0)), \mathtt{g}(n_0)^{\mathtt{r}(n_1)}, ..., \mathtt{g}(n_0)^{\mathtt{r}(n_4)}, \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)\mathtt{r}(n_\epsilon)} \sim \mathtt{G}(\mathtt{r}(n_0)), \mathtt{g}(\mathtt{r}(n_0)), \mathtt{g}(n_0)^{\mathtt{r}(n_1)}, ..., \mathtt{g}(n_0)^{\mathtt{r}(n_4)}, \mathtt{g}(n_0)^{\mathtt{r}(n)}.$$

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

$$\theta, \mathtt{G}(\mathtt{r}(n_0)), \mathtt{g}(\mathtt{r}(n_0)), \mathtt{g}(n_0)^{\mathtt{r}(n_1)}, ..., \mathtt{g}(n_0)^{\mathtt{r}(n_4)}, \mathtt{g}(n_0)^{\mathtt{r}(n_\delta)\mathtt{r}(n_\epsilon)} \sim \theta, \mathtt{G}(\mathtt{r}(n_0)), \mathtt{g}(\mathtt{r}(n_0)), \mathtt{g}(n_0)^{\mathtt{r}(n_1)}, ..., \mathtt{g}(n_0)^{\mathtt{r}(n_4)}, \mathtt{g}(n_0)^{\mathtt{r}(n)}$$

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 $\Phi(\mathsf{fold}(\Pi_2')) \sim \Phi(\mathsf{fold}(\Pi_2))$. This completes the proof. □

REMARK 2. 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 19, and for more parties the derivation is similar. □

9. DIGITAL SIGNATURES

In order 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 [20]). 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:

$\mathsf{k}(\_)$:	message	$\to$	message
$\mathsf{sign}(\_,\_)$:	message $\times$ message	$\to$	message
$\mathsf{ver}(\_,\_,\_)$:	message $\times$ message $\times$ message	$\to$	bool

Here $\mathsf{k}(\_)$ 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 $\mathsf{sk}(x) \stackrel{\text{def}}{\equiv} \pi_2(\mathsf{k}(n))$ are the public verification key and secret signing key parts of $\mathsf{k}(n)$ respectively. $\mathsf{sign}(y, z)$ is the message z signed with secret key y and $\mathsf{ver}(y, z, u)$ 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:
$$\mathsf{ver}\big(\mathsf{pk}(x), t, \mathsf{sign}(\mathsf{sk}(x), t)\big) = \textbf{\textit{true}}.$$
- *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 [20].
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 $\mathsf{sign}(\mathsf{sk}(n), t_1)$, $\mathsf{sign}(\mathsf{sk}(n), t_2)$, ..., $\mathsf{sign}(\mathsf{sk}(n), t_\ell)$. 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 $b_{t,u}^0, b_{t,u}^1, \ldots b_{t,u}^\ell$ be defined recursively as:
$$\begin{array}{lcl} b_{t,u}^0 & \stackrel{\text{def}}{\equiv} & \textbf{\textit{false}} \\ b_{t,u}^j & \stackrel{\text{def}}{\equiv} & \text{if } \mathsf{EQ}(t, t_j) \text{ then } \mathsf{ver}(\mathsf{pk}(n), t, u) \text{ else } b_{t,u}^{j-1} \end{array}$$
Then, the axiom schema is
$$\mathsf{ver}(\mathsf{pk}(n), t, u) = b_{t,u}^\ell.$$
That is, if t is one of t_j, the signature of which appears in t or u, then the RHS outputs $\mathsf{ver}(\mathsf{pk}(n), t, u)$. If t is neither of t_j, then the RHS 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 6. *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 $\mathcal{M}^c \not\models \mathsf{EQ}(\mathsf{ver}(\mathsf{pk}(n), t, u), b_{t,u}^\ell) \sim \textbf{\textit{true}}$ where $b_{t,u}^\ell$ 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

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

is a non-negligible function in η.

By definition, $[\![\textbf{\textit{true}}]\!]_{\rho,\eta} = 1$ and $[\![\mathsf{EQ}(\mathsf{ver}(\mathsf{pk}(n), t, u), b_{t,u}^\ell)]\!]_{\rho,\eta} = 1$ whenever $[\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} = [\![b_{t,u}^\ell]\!]_{\rho,\eta}$. Thus,

$$Adv^{\mathcal{A}}(\eta) \le \mathbf{Prob}\{\rho :\ [\![\mathsf{ver}(\mathsf{pk}(n), t, u)]\!]_{\rho,\eta} \ne [\![b_{t,u}^\ell]\!]_{\rho,\eta}\}.$$

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

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

Since $Adv^{\mathcal{A}}(\eta)$ is a non-negligible function in η, **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 $\mathsf{sign}(\mathsf{sk}(n), t_i)$, it consults the oracle. It is easy to see that the probability $\mathcal{B}$ wins the UF-CMA game is exactly **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 $f_0, f_1, ..., f_{\ell+1}, f'_{\ell+1} \in \mathcal{G}$. Let $t_1, t_2, \ldots, t_{\ell+1}, t, u$ be defined as follows:

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

Fix the interpretation of $f_0, f_1, ..., f_{\ell+1}, f'_{\ell+1}$ 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 i-th 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 j-th 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:

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

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, we have that

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

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 key, secret key pairs (pk_A, sk_A) and (pk_B, sk_B) are generated honestly for both A and B and pk_A, pk_B are 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 $\langle g^b, \mathsf{sign}(sk_B, \langle g^b, g^a\rangle)\rangle$, and computes $(g^a)^b$.
- The Initiator receives $\langle g^b, \mathsf{sign}(sk_B, \langle g^b, g^a\rangle)\rangle$, verifies the signature, computes $(g^b)^a$, and sends $\mathsf{sign}(sk_A, \langle g^a, g^b\rangle)$.
- The Responder receives $\mathsf{sign}(sk_A, \langle g^a, g^b\rangle)$, verifies the signature, and outputs `acc`.

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

Here we 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 authentication of the initiator, and the initiator's authentication of the responder can be handled similarly. 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. These assumptions can all be relaxed without the need of additional core axioms, but the formulation of the transition system becomes significantly more complex. We skip the definition of the transition system for lack of space.

Now, responder's authentication of the initiator means that if B received and verified a message that looked like $\mathsf{sign}(sk_A, \langle y, g^b\rangle)$ for some input y, then A has a matching session, which in this case just means that A has a session in which he sent $\mathsf{sign}(sk_A, \langle g^a, x\rangle)$ for some x before B received it, on the same branch (which is the same that A received $\langle x, \mathsf{sign}(sk_B, \langle x, g^a\rangle)\rangle$, and verification succeeded), and $\mathsf{EQ}(x, g^b)$ & $\mathsf{EQ}(y, g^a)$ 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 i as input, and if i 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:

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

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

PROPOSITION 7. *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 corresponding to the

condition $\mathsf{ver}(pk_A, \langle f_i(\phi_i), g^b\rangle, f_j(\phi_j))$ at the last move of the responder. Here $f_i(\phi_i)$ is the message that B received earlier and that was supposed to be g^a from A, while $f_j(\phi_j)$ is supposed to be $\mathsf{sign}(sk_A, \langle g^a, g^b\rangle)$ from A. According to UF−CMA and EQCONG, $\mathsf{ver}(pk_A, \langle f_i(\phi_i), g^b\rangle, f_j(\phi_j))$ 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 $\mathsf{sign}(sk_A, t)$ for some t such that $\mathsf{EQ}(\langle f_i(\phi_i), g^b\rangle, t)$ evaluates to true. By the role of A, $\mathsf{sign}(sk_A, t)$ must be of the form $\mathsf{sign}(sk_A, \langle g^a, f_h(\phi_h)\rangle)$ for an a that A generated at the beginning of the role in which this appears. That is, we are on the true side of the $\mathsf{EQ}(\langle f_i(\phi_i), g^b\rangle, \langle g^a, f_h(\phi_h)\rangle)$ branching. In the final oracle step, there is a branching according to $\mathsf{EQ}(f_i(\phi_i), g^a)$ & $\mathsf{EQ}(f_h(\phi_h), g^b)$, 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 $\mathsf{EQ}(\langle f_i(\phi_i), g^b\rangle, \langle g^a, f_h(\phi_h)\rangle)$, then in the final oracle step the branching $\mathsf{EQ}(f_i(\phi_i), g^a)$ & $\mathsf{EQ}(f_h(\phi_h), g^b)$ 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 $\mathsf{EQ}(f_i(\phi_i), g^a)$ &$\mathsf{EQ}(f_h(\phi_h), g^b)$ can be rewritten as

$$\mathsf{EQ}(\pi_1(\langle f_i(\phi_i), g^b\rangle), \pi_1(\langle g^a, f_h(\phi_h)\rangle))$$
$$\&\mathsf{EQ}(\pi_2(\langle g^a, f_h(\phi_h)\rangle), \pi_2(\langle f_i(\phi_i), g^b\rangle))$$

But by Example 11 and Example 12 if we are on the true side of $\mathsf{EQ}(\langle f_i(\phi_i), g^b\rangle, \langle g^a, f_h(\phi_h)\rangle)$, then this can be replaced by

$$\mathsf{EQ}(\pi_1(\langle g^a, f_h(\phi_h)\rangle), \pi_1(\langle g^a, f_h(\phi_h)\rangle))$$
$$\&\mathsf{EQ}(\pi_2(\langle g^a, f_h(\phi_h)\rangle), \pi_2(\langle g^a, f_h(\phi_h)\rangle))$$

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. □

11. 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. [11]) 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.

11.1 Encryptions

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

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

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

- ENC$_{\text{CPA}}$ if $\mathsf{sk}(n_1)$ does not occur anywhere,
- ENC$_{\text{CCA1}}$ if for any $t'[x]$ term with x explicitly occurring in $t'[x]$, the term $\mathsf{dec}(t'[x], \mathsf{sk}(n_1))$ is not a subterm of $\vec{t}[x]$, and
- ENC$_{\text{CCA2}}$ 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

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

  Formally, the formula above is ENC$_{\text{CCA2}}$ if each component term $t_i[x]$ of the vector $\vec{t}[x]$ is $(n_1, u, u') -$ ENC$_{CCA2}$ compliant as defined (recursively) below. We say that a term $t[x]$ is $(n, u, u') -$ 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 $t^1[x]$, ..., $t^r[x]$ such that $t[x] \equiv f(t^1[x], t^2[x], ..., t^r[x])$ and for any $t'[x]$ term containing x, $t[x] \not\equiv \mathsf{dec}(t'[x], \mathsf{sk}(n))$ and $t^i[x]$ is $(n, u, u') -$ ENC$_{CCA2}$ *compliant* for each $i = 1, \ldots, r$.

  - There are $(n, u, u') -$ ENC$_{CCA2}$ *compliant* terms $t'[x], t''[x]$, such that $t[x]$ is

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

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 $t_1[x]$ term containing x, then $t_1[x]$ is not the encryption oracle response, namely, x, and if it is, then the decryption returns $\mathbf{0}$. In fact, this definition of ENC$_{\text{CCA1}}$ is equivalent with the one in [7]

THEOREM 4. *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 ENC$_{\text{CCA2}}$, 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

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

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

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

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 an 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 $\mathrm{ENC}_{\mathrm{CCA1}}$ is the same, while the $\mathrm{ENC}_{\mathrm{CPA}}$ 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 $\mathrm{ENC}_{\mathrm{CCA2}}$ 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 $\mathrm{ENC}_{\mathrm{CCA1}}$ and $\mathrm{ENC}_{\mathrm{CPA}}$ 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 $\mathcal{B}$,

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

and

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

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,

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

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

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

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

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 m'th is the submission to the encryption oracle.

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

$$\begin{array}{lcl} t_1 & \stackrel{\mathrm{def}}{\equiv} & f_0(\mathsf{pk}(n)) \\ t_{i+1} & \stackrel{\mathrm{def}}{\equiv} & f_{i+1}(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), ..., \mathsf{dec}(t_i, \mathsf{sk}(n))) \\ u & \stackrel{\mathrm{def}}{\equiv} & f_m(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), ..., \mathsf{dec}(t_{m-1}, \mathsf{sk}(n))) \\ u' & \stackrel{\mathrm{def}}{\equiv} & f'_m(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), ..., \mathsf{dec}(t_{m-1}, \mathsf{sk}(n))) \\ u'' & \stackrel{\mathrm{def}}{\equiv} & f''_m(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), ..., \mathsf{dec}(t_{m-1}, \mathsf{sk}(n))) \\ t_{j+1}[x] & \stackrel{\mathrm{def}}{\equiv} & f_{j+1}(\mathsf{pk}(n), \mathsf{dec}(t_1, \mathsf{sk}(n)), ..., x, ..., c_j) \\ \vec{t}[x] & \stackrel{\mathrm{def}}{\equiv} & t_\ell[x] \end{array}$$

Where for $j = m, ..., \ell - 1$,

$$c_j = \begin{array}{l}\mathsf{if}\ \mathsf{EQ}(t_j[x], x) \wedge \mathsf{EQ}(\mathtt{L}(u), \mathtt{L}(u'))\ \mathsf{then}\ \mathbf{0}\\ \qquad\qquad\qquad\qquad\ \mathsf{else}\ \mathsf{dec}(t_j[x], \mathsf{sk}(n_1))\end{array}$$

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

$[\![f_0]\!]$ 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 i-th query to be submitted to the decryption oracle. Let s_i be the response or 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 $s_0, s_1, s_2, s_{m-1}, c, s_{m+1}, ..., s_{j-1}$ simulates the attacker $\mathcal{A}$ until it prepares the j-th 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,

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

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

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

$$\mathbf{Prob}\{\mathcal{A}(1^\eta,pk)=1\ \wedge\ b=0\}$$

and

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

$$\mathbf{Prob}\{\mathcal{A}(1^\eta,pk)=1\ \wedge\ b=1\}$$

But just as before,

$$\begin{array}{l}|\mathbf{Prob}\{\mathcal{A}(1^\eta,pk)=1\ \wedge\ b=1\}\\ \quad -\mathbf{Prob}\{\mathcal{A}(1^\eta,pk)=1\ \wedge\ b=0\})| =\\ |\mathbf{Prob}\{\mathcal{A}(1^\eta,pk)=b\}-\frac{1}{2}|\end{array}$$

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

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

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

and that completes the proof. □

EXAMPLE 21. Suppose that nonce and key generation are such that there are 0-ary function symbols ℓ_{nonce} and ℓ_{skey} such that $\mathtt{L}(n)=\ell_{\text{nonce}}$ and $\mathtt{L}(\mathsf{sk}(n))=\ell_{\text{skey}}$, and suppose also that pairing is length regular, that is, $\mathtt{L}(x_1)=\mathtt{L}(x_2)\wedge\mathtt{L}(y_1)=\mathtt{L}(y_2)\to\mathtt{L}(\langle x_1,x_2\rangle)=\mathtt{L}(\langle y_1,y_2\rangle)$. Note also that from the definition of the interpretation of $\mathtt{L}$, the formula $\mathtt{L}(\mathtt{L}(x))=\mathtt{L}(x)$ is sound. Consider real-or-random secrecy of n (let $\mathsf{k}_i\equiv\mathsf{k}(n_i)$, $\mathsf{r}_i\equiv\mathsf{r}(n_{i+2})$):

$$\begin{array}{l}\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2},\left\{\pi_2\Big(\mathsf{dec}\big(f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}),\mathsf{sk}_2\big)\Big),n\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1},n\\ \sim\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2},\left\{\pi_2\Big(\mathsf{dec}\big(f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}),\mathsf{sk}_2\big)\Big),n\right\}^{\mathsf{r}_2}_{\mathsf{pk}_1},n'\end{array}\quad(1)$$

with $f\in\mathcal{G}$. It is easy to show that the core axioms together with ENC_{CCA2} and the above properties of $\mathtt{L}$, 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 ENC_{CCA2} can be applied. For example, since dec acts on $f(...)$, we have to make sure that there is a conditioning as we required in the definition of ENC_{CCA2}.

So we start the proof by rewriting $f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})$ according to

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

by IFSAME, and then applying IFMORPH and the equations for encryption and pairing, we obtain

$$\pi_2(\mathsf{dec}(f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}),\mathsf{sk}_2))\overset{\text{EQCONG}}{=}$$

$$\pi_2\left(\mathsf{dec}\left(\begin{array}{l}\mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}),\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\\ \mathsf{then}\ f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\\ \mathsf{else}\ f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\end{array},\mathsf{sk}_2\right)\right)\overset{\text{Example 11}}{=}$$

$$\pi_2\left(\mathsf{dec}\left(\begin{array}{l}\mathsf{if}\ \mathsf{EQ}(f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}),\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\\ \mathsf{then}\ \{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}\\ \mathsf{else}\ f(\{\mathsf{sk}_1,n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2})\end{array},\mathsf{sk}_2\right)\right)\overset{\text{IFMORPH}}{=}$$

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

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

Let us define

$$\vec{t}[x]:=x,\left\{\begin{array}{l}\mathsf{if}\ \mathsf{EQ}(f(x),x)\\ \mathsf{then}\ n_5\\ \mathsf{else}\ \pi_2(\mathsf{dec}(f(x),\mathsf{sk}_2))\end{array}\right\}^{\mathsf{r}_1}_{\mathsf{pk}_2},n.$$

Note that $\vec{t}$ for k_2 satisfies the conditions for ENC_{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 ENC_{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 ENC_{CCA2}. Note then that the formula (1) is the same as

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

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

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

By the definition of $=$ and Example 10,

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

and so by IFTRUE,

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

and

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

Using this, EQCONG and ENC_{CCA2}, we get that

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

and

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

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

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

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

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

Hence, applying ENC_{CCA2} for a second time just as before, but now for k_1, we obtain that

$$\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \Big\{\pi_2\Big(\mathsf{dec}\big(f(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\big)\Big), n\Big\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n \sim \{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}, \Big\{\pi_2\Big(\mathsf{dec}\big(f(\{\mathsf{sk}(n_6), n_5\}^{\mathsf{r}_1}_{\mathsf{pk}_2}), \mathsf{sk}_2\big)\Big), n_7\Big\}^{\mathsf{r}_2}_{\mathsf{pk}_1}, n \tag{5}$$

Putting together formulas (2), (4), and (5), with TRANS and EQCONG, we have that

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

The same way we can derive that

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

But the right-hand sides of formulas (6) and (7) are equivalent as an immediate consequence of axioms FRESHIND and RESTR. Finally again transitivity delivers

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

which is what we wanted to show.

REMARK 3. *Example 21 illustrates the advantage of the BC technique for indistinguishability over the BC technique for reachability [6] 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 of in [8]. Here we could simply use* ENC_{CCA2} *and no new predicate was needed.*

11.2 Further Verification Results

Using axiom ENC_{CCA2}, we have also verified the NSL protocol for any number of sessions by hand: real-or-random secrecy of nonces shared between honest agents, authentication and agreement, agents being allowed to play both initiator and responder roles, even in matching sessions. These properties can be formulated for the NSL protocol with the help of oracles just as we saw for the two versions of the DH protocol. Then, just as we saw in Example 21, in each round of the protocol, we have to introduce if _ then _ else _ symbols to distinguish all cases when the decrypted adversarial message is any one of the previous honest encryptions, and when it is neither. When it is one of the previous encryptions, the decryption and the encryption cancel each other, and when the adversarial message is neither of the previous encryptions, then using ENC_{CCA2}, the plaintext can be replaced with 1's using L. Finally, the possibility that a given decryption is applied on the wrong honest encryption from another session or from the wrong message of the correct session has to be eliminated using axioms FRESHIND and FRESHNEQ. Without any further assumptions there are numerous attacks (not present in [23] because of assumptions on pairs and triples) such as that presented in [5]. However, if agents check whether bit strings expected to be nonces from the communicating parties have the correct length ℓ_{nonce} (see Example 21), then all these attacks are prevented.

The anonymity proof of the private authentication protocol presented in [7] is also easily generalized.

12. CONCLUSIONS

We have introduced key extensions to the core of computationally complete symbolic attacker based on indistinguishability first introduced in [7] that are necessary to apply the technique to analyze protocols allowing multiple sessions. Towards 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 NSL protocol and the verification of authentication of a simplified version of the STS protocol and authentication and agreement for the NSL protocol, we showed how the model can be used to tackle multiple sessions, algebraic properties, real-or-random secrecy, and even trace properties.

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 [22] for verification of reachability properties in the BC framework. For reachability, verification for a large class of protocols turns out to be decidable in co-NP [22]. Note as well that PROVERIF is not guaranteed to terminate, but still can be used for verification very efficiently.

13. REFERENCES

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL'01*, pages 104–115. ACM, 2001.

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

[3] M. Abdalla, P-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC'05*, LNCS, pages 65–84, 2005.

[4] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS'03*, pages 220–230. ACM, 2003.

[5] G. Bana, P. Adão, and H. Sakurada. Computationally Comlete Symbolic Attacker in Action. In *FSTTCS'12*, LIPIcs, pages 546–560. Schloss Dagstuhl, 2012.

[6] G. Bana and H. Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *POST'12*, LNCS, pages 189–208. Springer, 2012.

[7] G. Bana and H. Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In *CCS '14*, pages 609–620. ACM, 2014.

[8] G. Bana, K. Hasebe, and M. Okada. Computationally complete symbolic attacker and key exchange. In *CCS '13*, pages 1231–1246. ACM, 2013.

[9] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. Cryptology ePrint Archive, Report 2015/074, 2015. http://eprint.iacr.org/.

[10] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, 2011.

[11] M. Bellare, A. Desai, D. Pointcheval, and Ph. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO'98*, LNCS. Springer, 1998.

[12] B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *CADE'05*, Tallinn, Estonia, July 2005.

[13] B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.

[14] D. Boneh. The decision diffie-hellman problem. In *ANTS-III'98*, pages 48–63. Springer-Verlag, 1998.

[15] V. Cheval and B. Blanchet. Proving more observational equivalences with proverif. In *POST'13*, Lecture Notes in Computer Science, pages 226–246. Springer, 2013.

[16] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *CCS'08*, pages 109–118. ACM, 2008.

[17] C. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *CAV'08*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008.

[18] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *CAV'05*, volume 3576 of *LNCS*, pages 281–285, 2005.

[19] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *FMSE'05*, pages 23–32. ACM, 2005.

[20] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.

[21] R. Küsters and M. Tuengerthal. Computational soundness for key exchange protocols with symmetric encryption. In *CCS'09*, pages 91–100. ACM, 2009.

[22] Guillaume Scerri. *Proofs of security protocols revisited*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, January 2015.

[23] Bogdan Warinschi. A computational analysis of the needham-schroeder protocol. In *16th Computer security foundation workshop (CSFW)*, pages 248–262. IEEE, 2003.