FORMAL PROOFS OF CRYPTOGRAPHIC SECURITY OF NETWORK PROTOCOLS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Arnab Roy

December 2009

Abstract

Present-day internet users and networked enterprises rely on key management and related protocols that use cryptographic primitives. In spite of the staggering financial value of, say, the total number of credit card numbers transmitted by SSL/TLS in a day, we do not have correctness proofs that respect cryptographic notions of security for many of these relatively simple distributed programs. In light of this challenge, there have been many efforts to develop and use methods for proving security properties of network protocols. Computational Protocol Composition Logic (CPCL), developed by our group at Stanford, is a symbolic logic whose semantics is defined with respect to the complexity-theoretic model of cryptography. The axiomatic proofs in CPCL do not involve probability and complexity and are amenable to automation. Furthermore, the soundness theorem guarantees that they provide comparable mathematical guarantees as traditional hand-proofs done by cryptographers.

Protocol authentication properties are generally trace-based, meaning that authentication holds for the protocol if authentication holds for individual traces (runs of the protocol and adversary). Computational secrecy conditions, on the other hand, often are not trace based: the ability to computationally distinguish a system that transmits a secret from one that does not, is measured by overall success on the *set* of all traces of each system. Non-trace-based properties present a challenge for inductive or compositional methods: induction is a natural way of reasoning about traces of a system, but it does not appear directly applicable to non-trace properties. We therefore investigate the semantic connection between trace properties that could be established by induction and non-trace-based security requirements.

In this dissertation, we present foundations for inductive analysis of computational security properties by proving connections between selected trace properties of protocol executions and non-trace complexity theoretic properties standard in the literature. Specifically, we prove that a certain trace property implies computational secrecy and authentication properties, assuming the encryption scheme provides chosen ciphertext security and ciphertext integrity. We formalize the aforesaid inductive properties in a set of new axioms and inference rules that are added to CPCL and prove soundness of the system over a standard cryptographic model with a probabilistic polynomial time adversary. We illustrate the system by giving a modular, formal proof of computational authentication and secrecy properties of Kerberos V5.

We also present axioms and inference rules for reasoning about Diffie-Hellman-based key exchange protocols and use these rules to prove authentication and secrecy properties of two important protocol standards, the Diffie-Hellman variant of Kerberos, and IKEv2, the revised standard key management protocol for IPSEC. The proof system extended with the new axioms and rules is sound for an accepted semantics used in cryptographic studies. In the process of applying our system, we uncover a deficiency in Diffie-Hellman Kerberos that is easily repaired.

Acknowledgements

This Dissertation is the culmination of some wonderful years at Stanford. It brings together the central pieces of research that I did with my group. Yet the most significant impact of these years will remain unwritten - research as a way of life rather than work. For that I am most thankful to John C. Mitchell, who has not only been my Doctoral Adviser but also a great friend. These years at Stanford may be short in the context of a research career, but substantial enough to have a plethora of experiences and early enough to be formative. At all times I have found a supporting person in John and looked up to him as an inspiration.

Working with my other principal collaborators Anupam Datta and Ante Derek has been an immense pleasure. Anupam has also been a mentor to me and his experiences as a researcher at the beginning of his career have been very valuable advice. I also had fun working with Mukund Sundararajan on diverse topics.

I would also like to thank my past mentors who shaped my perception of research in substantial ways - Partha P. Chakrabarti and Rajeev Kumar at IIT Kharagpur, Jean-Pierre Seifert at Intel Corporation, Yuri Gurevich at Microsoft Research, Suresh Chari and Charanjit Jutla at IBM Research.

Last but not the least, I would like to thank my parents Subhash Chandra Roy and Bandana Roy for showing me the light of the world and providing me this gift of education. Without their loving support I could not have come to the threshold of this wonderful career.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Present-day internet users and networked enterprises rely on key management and related protocols that use cryptographic primitives. In spite of the staggering financial value of, say, the total number of credit card numbers transmitted by SSL/TLS in a day, we do not have correctness proofs that respect cryptographic notions of security for many of these relatively simple distributed programs. In light of this challenge, there have been many efforts to develop and use methods for proving security properties of network protocols. This area of research has had two important but historically independent foundations, one based on logic and symbolic computation, and one based on computational complexity theory. The symbolic approach, which uses a highly idealized representation of cryptographic primitives, has been a successful basis for formal logics and automated tools. Conversely, the computational approach yields more insight into the strength and vulnerabilities of protocols, but is more difficult to apply.

1.1 Contributions

Protocol Composition Logic (PCL) is a logic we have developed for proving security properties of network protocols in the symbolic model of cryptography. The logic is designed around a process calculus with actions for possible protocol steps including generating new random numbers, sending and receiving messages, and performing decryption and digital signature verification actions. The proof system consists of axioms about individual protocol actions and inference rules that yield assertions about protocols composed of multiple steps. Although assertions are written only using the steps of the protocol, the logic is sound in a strong sense: each provable assertion involving a sequence of actions holds in any protocol run containing the given actions and arbitrary additional actions by a malicious adversary. This approach lets us prove security properties of protocols under attack while reasoning only about the actions of honest parties in the protocol. PCL supports compositional reasoning about complex security protocols and we have applied it to a number of industry standards including SSL/TLS, IEEE 802.11i and Kerberos V5.

Computational PCL evolved out of PCL retaining similar syntax and proof systems, but the proofs now guaranteeing security in the computational model of cryptography. In both the symbolic and computational models of network protocol execution and attack, a protocol defines a set of possible traces. In the computational model, the set of traces is indexed by a security parameter, and there is a probability distribution arising from randomness used in cryptographic actions in the protocol and randomness used by the protocol adversary. Some properties of a protocol are trace properties, meaning that the property can be observed to hold or fail on an individual trace, and the property holds for a set of traces iff it holds for all but a negligible subset. However, many natural secrecy conditions, in the computational model using probabilistic polynomial-time computation, are not trace based. Computational indistinguishability, for example, requires that no computational observer can feasibly distinguish a situation in which a secret is transmitted from a situation in which some non-informative values are transmitted instead. If we look at a single trace, this gives no real information about how likely an observer is to succeed. Instead, we must look at the probability distribution on traces, and determine the probability of success over the entire distribution. The nature of this property presents a challenge for proving computational secrecy properties of protocols, since trace-based properties are naturally amenable to induction, while non-trace-based properties are not.

As part of this dissertation, we present the following theoretical contributions to the security analysis of network protocols:

1. We develop foundations for inductive analysis of computational security properties by proving connections between selected trace properties of protocol executions and non-trace complexity theoretic properties standard in the literature.

2. We formalize the aforesaid inductive properties in a set of new axioms and inference rules that are added to Computational PCL and proved soundness of the system over a standard cryptographic model with a probabilistic polynomial time adversary.

On the practical side, we apply the above techniques to the analysis of real world network security protocols:

1. We prove authentication and secrecy properties of the Kerberos V5 protocol with both symmetric-key and public-key initialization in the complexity theoretic models. The proofs of key secrecy of Kerberos V5 in the complexity theoretic model are first in the literature.

2. We discover a deficiency in Kerberos V5 with Diffie-Hellman initialization and suggest simple corrective measures. For the first time in the literature, we prove secrecy and authentication properties of this mode of Kerberos in the complexity theoretic model, thus constructing proofs of security of all three modes of Kerberos V5 in a uniform formal framework.

3. We prove secrecy and authentication properties of the IKEv2 protocol in the complexity theoretic model for the first time in the literature.

1.2 Related Work

Most demonstrated approaches for proving security of complex network protocols, of the scale that appear in IEEE and IETF standards, use a simplified model of protocol execution based on symbolic computation and highly idealized cryptography [12, 22, 27, 33]. However, proofs about symbolic computation do not provide the same level of assurance as proofs about probabilistic polynomial-time attacks. Several groups of researchers have therefore developed methods for deriving cryptographic meaning from properties of symbolic protocol execution [9, 5, 26, 30, 49, 50, 58].

One class of methods involve showing that the behavior of a symbolic abstraction, under symbolic attacks, yields the same significant failures as a finer-grained execution under finer-grained probabilistic polynomial-time attack. However, in such equivalence theorems there are no known suitable symbolic abstractions of Diffie-Hellman exponentiation. Specifically, in other studies of Diffie Hellman key exchange, [48] uses a symbolic model, while [53] imposes non-standard protocol assumptions. In addition, there are theoretical negative results in the setting of *universal composability* or *reactive simulatability* that suggest that correspondence with ideal functionality may be impossible for symmetric encryption if a protocol might reveal a secret key [25, 31], or for hash functions or exclusive-or [8, 10]. While the assumptions about the cryptographic primitives in [30] are similar to those in our work for encryption, they do not allow secrets to be used as keys and also do not consider Diffie-Hellman key exchange.

In contrast, Computational PCL supports direct reasoning about properties of probabilistic polynomial-time execution of protocols, under attack by a probabilistic polynomial-time adversary, without explicit formal reasoning about the adversary, probability or complexity. In addition, different axioms may depend on different cryptographic assumptions, allowing us to consider which assumptions are actually necessary for each property we establish. Prior work on Computational PCL describes the core logic and semantics [35], and was used to study protocol composition and key exchange [36]. However, this dissertation describes the first presentation of applicable general semantic results about non-trace secrecy properties.

More generally, Abadi and Rogaway [2] initiated computationally sound symbolic analysis of static equivalence, with extensions and completeness explored in [57, 3]; a recent extension to Diffie-Hellman appears in [21], covering only *passive adversaries* (as in prior work in this line), not the stronger active adversaries used in this dissertation. A language-based approach to computational protocol analysis is taken by Mitchell et al [54, 60, 65, 61]. They develop a process calculus for expressing probabilistic polynomial time protocols, a specification method based on a compositional form of equivalence, and an equational reasoning system for establishing equivalence between processes. In subsequent work, Blanchet *et al* have developed and implemented additional proof techniques in a tool called CryptoVerif [18, 19], which uses equivalences and a probabilistic polynomial-time process calculus inspired by pi-calculus and [54, 60, 65, 61].

1.3 Organization

This dissertation is organized as follows: In Chapter 2 we describe the mathematical foundations we developed to inductively reason about protocol actions. In Chapter 3 we formalize this inductive reasoning, for protocols using public and symmetric key encryptions, in a set of new axioms and inference rules that are added to Computational PCL. We apply the proof system to Kerberos V5 with symmetric and public-key initialization. In Chapter 4 we extend the formalization of this inductive reasoning for protocols using Diffie-Hellman key exchange. We apply the proof system to Kerberos V5 with Diffie-Hellman initialization and IKEv2. We conclude and speculate future directions in Chapter 5.

Chapter 2

Inductive Trace Properties for Computational Security

In symbolic and computational models of network protocol execution and attack, a protocol and adversary define a set of possible traces (runs). In the computational model, which we consider in this dissertation, there is a set of traces for each value of the security parameter, and there is a probability distribution on each such set arising from randomness used in cryptographic actions in the protocol and randomness used by the protocol adversary. Some properties of a protocol are trace properties, meaning that the property can be observed to hold or fail on an individual trace, and the property holds for a set of traces iff it holds for all but a negligible subset. For example, the authentication property "if Bob accepts a key for use with Alice, then Alice participated in the same session of the same key generation protocol" is a trace property. We can see, for any given trace, whether Bob accepted the key and whether Alice participated in the same session of the same protocol. However, many natural secrecy conditions in the computational model are not trace based.

Computational indistinguishability, for example, requires that no computational observer can feasibly distinguish a situation in which a secret is transmitted from a situation in which some non-informative values are transmitted instead. If we look at a single trace, this gives no real information about how likely an observer is to succeed over the set of all traces. Instead, we must look at the probability distribution on traces, and determine the probability of success over the entire distribution. Computational indistinguishability and other non-trace properties present a challenge for proving secrecy properties of protocols, since trace-based properties are naturally amenable to induction on the length of a trace, while non-trace-based properties are not. If we assume inductively that a trace-based property holds, this means it holds for (almost) all traces, and we can consider the effect of adding one more step to each trace. If the effect preserves the property on each trace, then we conclude that the property holds for the protocol as a whole. Since this form of argument only works for trace-based properties, it does not appear applicable to important computational security properties.

In this chapter, we develop foundations for inductive proofs of computational security properties by proving connections between selected trace properties and useful non-trace properties. In Chapters 3 and 4, we use the results established here to extend Computational Protocol Composition Logic (Computational PCL) [35, 36, 66, 69] to computational secrecy properties, and use that logic to prove properties of standard and useful protocols.

We say that a protocol is *secretive* if the protocol participants protect values intended to be kept secret in certain ways. While one cannot immediately see syntactically whether a protocol is secretive or not, this is a trace property because it involves conditions on individual actions by honest parties to a protocol. We prove that all secretive protocols have computational secrecy and authentication properties, assuming the encryption scheme used provides chosen ciphertext security and ciphertext integrity. In addition, we prove a similar theorem for computational secrecy assuming Decisional Diffie-Hellman and a chosen plaintext secure encryption scheme. The computational security guarantees for secretive protocols are established first for the simple case when secrets are protected by pre-shared "level-0" keys (Theorems 1-3), then generalized (Theorems 4-6) under the condition that each key is protected by predecessor keys in an acyclic graph. This condition avoids the difficulty of dealing with key cycles while assuming only IND-CCA security of encryption. However, the properties we are able to prove do depend on key use, as should be expected in light of previous results [30, 45, 36, 64]. Specifically, we prove strong key indistinguishability properties for a class of secretive protocols that do not use the established secret as a key (Theorems 1, 4), and a weaker key usability property for secretive protocols that allows the established secret to be used as a key (Theorems 2, 5, 7).

The key proof technique used in obtaining these results is the "bilateral simulator", which consistently maintains two possible values of the secret in carrying out the simulation. A similar simulator is used by Micciancio and Warinschi in their first result on a correspondence theorem between symbolic and computational trace properties [58] and much subsequent work by others (e.g. [30]). One key difference is the degree of formalization in our work. While previous papers described the behavior of the simulator without a precise step-by-step specification, we present an inductive definition of the operational behavior of the simulator and prove that it behaves correctly (see Section 2.1.4). The correctness proof for the simulator is non-trivial and justifies the effort. This method can be used to formalize the simulators used in related work. Another difference from prior work is that we allow for the use of secrets to be used as keys. In this situation, key indistinguishability does not hold, but key usability (introduced by Datta et al [36]) still holds. This is important for applications because most practical protocols distribute secrets and then use them as keys. In addition, we prove computational secrecy properties even when the secret keys are protected by predecessor keys in an acyclic graph. This is particularly relevant for protocols like Kerberos where key distribution proceeds in multiple stages by encrypting new keys under previously generated keys. Finally, we provide guarantees about secrets set up using Diffie-Hellman key exchange, subject to certain constraints.

Section 2.1 describes the protocol programming language, computational execution model and security properties. A trace-based definition of "secretive protocols" and associated computational security theorems (Theorems 1–6) are presented in section 2.1.2. A similar trace-based definition of protocols that use the Diffie-Hellman primitive safely and associated computational secrecy theorem (Theorem 7) is presented in section 2.2.

2.1 Computational Model

2.1.1 Modeling Protocols

We use a simple protocol programming language to present a *protocol* as a set of programs, one for each *role* such as “Initiator”, “Responder” or “Server”. Each role program is a sequence of protocol actions to be executed by an honest participant (see [40, 32, 33, 34] for the syntax and operational semantics of the protocol language). The protocol actions, which include nonce generation, symmetric encryption and decryption, and communication steps (sending and receiving), are given in Table 2.1. A principal executing an instance of a role is called a *thread*. A principal can simultaneously execute multiple threads. Symmetric encryption with a nonce as a key signifies encryption with a key deterministically generated from the nonce. It is possible to naturally extend the results of this chapter to asymmetric encryption and signature, but for simplicity of exposition we do not present those extensions.

We consider an essentially standard two-phase execution model as in [16], with static rather than adaptive corruption. The execution environment has an infinite set of principals, a subset of which are labeled honest and the rest are dishonest. Every principal has access to an infinite sequence of random coins and every pair of principals have a shared symmetric key. In the execution phase, the adversary executes the protocol by interacting with the principals. The adversary can ask for the keys of the dishonest principals whereas it is not allowed to do so for the honest principals. We make the standard assumption that the adversary has complete control over the network, i.e., it sends messages to the parties and intercepts their answers, as in the accepted cryptographic model of [16]. The adversary also triggers the creation of new threads. The length of keys, nonces, etc. as well as the running time of the protocol parties and the attacker are polynomially bounded in the security parameter. Note that due to the computational limitations of the adversary, it will only be able to interact with a polynomially bounded number of principals.

Informally, a *trace* is a record of all actions executed by honest principals and the attacker during protocol execution. Since honest principals execute symbolic programs, a trace contains symbolic descriptions of the actions executed by honest

(keys)	K	$::=$	k	pre-shared symmetric key
			n	nonce
(atomic terms)	u	$::=$	x	atomic term variable
			K	keys
			N	name
			M	data payload
(terms)	t	$::=$	y	term variable
			u	atomic term
			$t.t$	tuple of terms
			$ENC[K](t)$	term encrypted with key K
(actions)	a	$::=$	`send` t	send a term t
			`receive` y	receive term into variable y
			`new` n	generate nonce n
			`match` t `as` t	match a term to a pattern
			$y :=$ `pair` t, t	pair terms
			$y :=$ `fst` t	first component of a pair
			$y :=$ `snd` t	second component of a pair
			$y :=$ `enc` t, K	encrypt term
			$y :=$ `dec` t, K	decrypt term
(program)	P	$::=$	$a;$	single action
			$Pa;$	sequence of actions
(thread)	X	$::=$	(P, N)	program executed by N

Table 2.1: Syntax of the Protocol Programming Language - terms and actions

parties as well as the mapping of bitstrings to variables. On the other hand, since the attacker is an arbitrary probabilistic poly-time algorithm, the trace only records the send-receive actions of the attacker (and the corresponding mapping to bitstrings), but not internal actions of the adversary.

More formally, a trace is a pair $\langle e, \lambda \rangle$, where e records the symbolic actions of protocol participants and send/receive actions of attacker, and λ maps symbolic terms in actions to bitstrings using appropriate functions. For example, e may indicate that a thread sends $ENC[k](t)$, a message consisting of a term t encrypted with key k, and λ gives the bitstring values of variables in t, the key k and the encryption randomness. The adversary triggers the creation of new threads belonging to the given principals. Every action in a trace is an action of a specific thread, which is identified by a principal and a thread ID (to make different threads of the same principal unique), or an attacker action. Actions associated with a thread of the protocol have a symbolic representation, because protocols are written symbolically, but attacker actions are carried out by an arbitrary probabilistic polynomial-time algorithm and only appear symbolically in the form of bit-string values received from the network by symbolic traces. If the symbolic action involves some thread generating a new nonce s, then $\lambda(s)$ is the bitstring obtained by applying a nonce-generation algorithm (which uses the random coins available to that thread). Similarly, symbolic symmetric encryption terms are mapped to bitstrings obtained by applying an encryption function to the bitstring representation of the corresponding plaintext term given by λ. The computational interpretation of decryption is defined similarly.

2.1.2 Modeling Security Properties

Authentication and integrity are generally trace properties. In this chapter, we focus on simple integrity properties of the form that a certain encrypted message was produced by a specific principal. Such a property is satisfied by a protocol if for all probabilistic poly-time attackers and sufficiently large security parameters this property holds in almost all runs of the protocol, where “almost all” means all but a negligible fraction of the runs, as is standard in cryptographic studies [16]. The condition "almost all" allows for the fact that a desired property may fail in very unlikely situations such when the attacker manages to guess all the bits of a key. The interested reader is referred to [35] for a formal definition.

Computational secrecy is a more subtle property. It is a property of a set of traces and not a single trace. We consider two notions of computational secrecy—one based on the standard cryptographic notion of *indistinguishability* and the other called *key usability*, first presented in [36]. We describe some problems with inductive reasoning about key indistinguishability and discuss the alternative condition that appears more suitable for our purposes.

We summarize our broad assumptions here:

- The execution model only has static corruption.
- Bitstrings corresponding to (pairs, encryption, ...) are assumed to be type tagged - we will explain this technically in section 2.1.4.
- For simplicity, the only cryptographic primitives we consider in this chapter are symmetric encryption (with atomic keys) (Section 2.1.2) and Diffie-Hellman (Section 2.2). As we will see in the subsequent chapters, we have also worked with protocols having public-key encryption, signatures and hashes and the extensions have been natural.

Basic Cryptographic Definitions

This section provides standard cryptographic definitions for reference (see [17] for additional details).

Definition 1 (Symmetric Encryption Scheme) *A symmetric encryption scheme $\mathcal{ES} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:*

- *The randomized key generation algorithm $\mathcal{KG}$ takes as input the security parameter η (in unary) and returns a string k. We let $Keys(\mathcal{ES})$ denote the set of all strings that have non-zero probability of being output by $\mathcal{KG}$. The members of this set are called keys. We write $k \leftarrow \mathcal{KG}(\eta)$ for the operation of executing $\mathcal{KG}$ and letting k denote the key returned.*

- *The encryption algorithm $\mathcal{E}$, which might be randomized or stateful, takes a key $k \in Keys(\mathcal{ES})$ and a plaintext $M \in \{0,1\}^*$ to return a ciphertext $C \in \{0,1\}^* \cup \{\perp\}$, denoted $\mathcal{E}_k(M)$.*

- *The deterministic decryption algorithm $\mathcal{D}$ takes a key $k \in Keys(\mathcal{ES})$ and a ciphertext $C \in \{0,1\}^* \cup \{\perp\}$ to return some $M \in \{0,1\}^*$, denoted $\mathcal{D}_k(C)$.*

  The scheme is said to provide correct decryption if for any key $k \in Keys(\mathcal{ES})$, any message $M \in \{0,1\}^*$, and any ciphertext $\mathcal{E}_k(M)$ obtained by encrypting this message, it is the case that $\mathcal{D}_k(\mathcal{E}_k(M)) = M$.

Definition 2 (LR Oracle) *Let $\mathcal{ES} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. The left-or-right encryption oracle (LR oracle) for $\mathcal{ES}$ is defined as follows, given $b \in \{0,1\}$ and $M_0, M_1 \in \{0,1\}^*$:*

```
Oracle $\mathcal{E}_k(LR(M_0, M_1, b))$
    if $|M_0| \neq |M_1|$ then return $\perp$
    else return $\mathcal{E}_k(M_b)$
```

Definition 3 (IND-CPA) *The experiment IND-CPA (Indistinguishability under Chosen Plaintext Attack), for adversary A, is defined as:*

```
Experiment $\mathbf{Exp}^b_{IND\text{-}CPA,A}(\eta)$
    $k \leftarrow \mathcal{KG}(\eta)$
    $d \leftarrow A^{\mathcal{E}_k(LR(\cdot,\cdot,b))}(\eta)$
    return $d$
```

A query to any LR oracle consists of two messages of equal *length. The* advantage *of A is defined as:*

$$\mathbf{Adv}_{IND\text{-}CPA,\mathcal{A}}(\eta) = \Pr[\mathbf{Exp}^0_{IND\text{-}CPA,\mathcal{A}}(\eta) = 0] - \Pr[\mathbf{Exp}^1_{IND\text{-}CPA,\mathcal{A}}(\eta) = 0]$$

The encryption scheme $\mathcal{ES}$ is IND-CPA *secure if the advantage of any probabilistic poly-time adversary $\mathcal{A}$ is negligible in the security parameter.*

Definition 4 (IND-CCA) [1] *The experiment IND-CCA (Indistinguishability under Chosen Ciphertext Attack), for adversary $\mathcal{A}$, is defined as:*

```
Experiment $\mathbf{Exp}^{b}_{\textit{IND-CCA},\mathcal{A}}(\eta)$
    $k \leftarrow \mathcal{KG}(\eta)$
    $d \leftarrow \mathcal{A}^{\mathcal{E}_k(LR(\cdot,\cdot,b)),\mathcal{D}_k(\cdot)}(\eta)$
    return $d$
```

A query to any LR oracle consists of two messages of equal *length and that adversary $\mathcal{A}$ does not query $\mathcal{D}_k(\cdot)$ on an output of $\mathcal{E}_k(LR(\cdot,\cdot,b))$. The* advantage *of $\mathcal{A}$ is defined as:*

$$\mathbf{Adv}_{\textit{IND-CCA},\mathcal{A}}(\eta) = \Pr[\mathbf{Exp}^{0}_{\textit{IND-CCA},\mathcal{A}}(\eta) = 0] - \Pr[\mathbf{Exp}^{1}_{\textit{IND-CCA},\mathcal{A}}(\eta) = 0]$$

The encryption scheme $\mathcal{ES}$ is IND-CCA *secure if the advantage of any probabilistic poly-time adversary $\mathcal{A}$ is negligible in the security parameter.*

Definition 5 (INT-CTXT) *The experiment INT-CTXT (Ciphertext Integrity), for adversary $\mathcal{A}$, is defined as:*

```
Experiment $\mathbf{Exp}_{\textit{INT-CTXT},\mathcal{A}}(\eta)$
    $k \leftarrow \mathcal{KG}(\eta)$
    $c \leftarrow \mathcal{A}^{\mathcal{E}_k(\cdot)}(\eta)$
    return $c$
```

The output c of adversary $\mathcal{A}$ should not have been a response of $\mathcal{E}_k(\cdot)$. The advantage *of $\mathcal{A}$ is defined as:*

$$\mathbf{Adv}_{\textit{INT-CTXT},\mathcal{A}}(\eta) = \Pr[\mathbf{Exp}_{\textit{INT-CTXT},\mathcal{A}}(\eta) \textit{ can be decrypted successfully with key } k]$$

The encryption scheme $\mathcal{ES}$ is INT-CTXT *secure if the advantage of any probabilistic poly-time adversary $\mathcal{A}$ is negligible in the security parameter.*

Definition 6 (DDH) *A group family $\mathcal{G}$ is a set of finite cyclic groups $\mathcal{G} = \{G_\lambda\}$,*

[1] In the sequel, we use an extension of IND-CCA to the multi-user setting with $|\mathcal{K}|$ keys, following [13]. We refer to this definition as $|\mathcal{K}|$-IND-CCA.

where λ ranges over an infinite indexed set. Let η be a security parameter. An instance generator IG for $\mathcal{G}$ is a probabilistic polynomial time algorithm (in η) that outputs an index λ and a generator g of G_λ. The Decisional Diffie-Hellman (DDH) assumption states that for all probabilistic polynomial time adversaries $\mathcal{A}$, every constant α and all sufficiently large η's, we have:

$$\left|\Pr[\mathcal{A}(\lambda, g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(\lambda, g, g^a, g^b, g^c) = 1]\right| < 1/\eta^\alpha$$

Here the probabilities are taken over the random bits of $\mathcal{A}$, the choice of $\langle\lambda, g\rangle$ according to the distribution $IG(1^\eta)$, and the choice of a, b, and c uniformly at random in $[1, \mid G_\lambda \mid]$.

Key indistinguishability

Intuitively, key indistinguishability means that an attacker cannot distinguish the actual key produced by a run of the protocol from a random key drawn from the same distribution. In [11] the secrecy requirement for authenticated key exchange protocols is defined by issuing a random challenge to the adversary, while the corresponding notion in [9] allows an adversary to compare the run of a protocol to a simulated ideal protocol. As in [29], we define key indistinguishability using a cryptographic game that is similar to [11] but adapted to the way our protocol execution model is formulated. The game involves a two-phase adversary $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$. In the *key exchange phase*, the honest parties run sessions of the protocol following the execution model described in Section 2.1.1. At the end of the key exchange phase, the adversary selects a challenge session among all sessions executed by the honest parties, and outputs some state information representing the information $\mathcal{A}_e$ was able to gather during its execution. It does not matter if A_e is interacting with dishonest principals as well, as the action of dishonest principals can be simulated by A_e itself. We therefore assume $\mathcal{A}_e$ is interacting only with honest principals. Let k be the key locally output by the honest party executing the session. At this point, the experiment enters its second phase—the *challenge phase* where the goal of the adversary $\mathcal{A}_c$ is to distinguish the key k from a random key r drawn from the same distribution using the state information previously output by $\mathcal{A}_e$. The protocol is said to satisfy key indistinguishability if the success probability of $\mathcal{A}_c$ is bounded above 1/2 by a negligible function of the security parameter.

Key indistinguishability turns out to be too strong a condition in many practical scenarios. Specifically, even if a key exchange protocol run in isolation satisfies this condition, key indistinguishability is generally lost as soon as the key is used to encrypt a message of a known form or with partially known possible content. Moreover, some situations allow one agent to begin transmitting encrypted data before the other agent finishes the last step of the key exchange, rendering key indistinguishability false at the point that the key exchange protocol finishes. This appears to be the case for SSL [42]; see [59] for a discussion of data transfer before the key exchange *finished* messages are received. Furthermore, some key exchange protocols even use the generated key during the protocol, preventing key indistinguishability. Key indistinguishability may not hold in these cases even if the encryption scheme is key concealing. Fortunately, many protocols that use keys do not require key indistinguishability to provide meaningful security guarantees. In particular, semantic security [44] does *not* require that the keys used remain indistinguishable from random. To circumvent the technical problems we encountered in working with key indistinguishability, we developed an alternative notion, *key usability*, in [36] that is parameterized by the security goal of the application in which the resulting key is used.

Rackoff makes a distinction between stronger and weaker notions of distinguishability, illustrated by example in the appendix of [24]. The basic idea is that an adversary who can continue to interact with the protocol after a challenge has been issued may have more power than an attacker who cannot. This leads to a distinction between key indistinguishability based on an attacker who uses a challenge (the key or a surrogate chosen randomly from the same distribution) to interact with subsequent steps of the key exchange protocol, and indistinguishability based on an attacker who cannot execute further protocol steps. The definition of key usability that we use in this dissertation is similar to the weaker notion of key indistinguishability in that the adversary who attempts to win, for example, the IND-CPA game for encryption, does not have the opportunity to interact further with other protocol participants. On the other hand, because protocols (such as SSL [42]) that provide key confirmation steps will also fail the stronger form of definition suggested by Rackoff, we consider the weaker condition we use advantageous for certain practical settings. Specifically, different protocols in current use achieve different properties, and there is value to stating and proving these properties precisely. We also hope that the setting presented in this dissertation provides a useful starting point for expressing and reasoning about stronger security conditions.

Key usability

We define usability of keys obtained through a key exchange protocol Σ with respect to a class of applications S via a two-phase experiment. The experiment involves a two-phase adversary $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$. In the *key exchange phase*, the honest parties run sessions of the protocol following the standard execution model. At the end of the key exchange phase, the adversary selects a challenge session among all sessions executed by the honest parties, and outputs some state information representing the information $\mathcal{A}_e$ was able to gather during its execution. Let k be the key locally output by the honest party executing the session. At this point, the experiment enters its second phase—the *challenge phase* where the goal of the adversary is to demonstrate an attack against a scheme $\Pi \in S$ which uses the key k. After $\mathcal{A}_e$ receives as input St, it starts interacting with Π according to the game used for defining security of the application protocols in S. For example, if S is a set of encryption schemes, then the relevant game may be IND-CPA or IND-CCA [43].

We formalize the case when the game defines IND-CPA security. $\mathcal{A}_c$ has access to a left-right encryption oracle under k, and in addition, it receives as input the state information from $\mathcal{A}_e$. The advantage of the adversary is defined as for the standard IND-CPA game with the difference that the probability is taken over the random coins of the honest parties (used in the execution of the protocol), the coins of the two adversaries, and the coins used for encryption in the challenge phase. The keys obtained by running the key exchange protocol are usable for the schemes in S if this advantage is bounded above by a negligible function of the security parameter, for *all* encryption schemes in S. The universal quantification over schemes is used to capture the fact that the security property is guaranteed for all encryption schemes which satisfy the IND-CPA condition. The definition can be easily modified to define a similar usability property of keys for other primitives, for example, message authentication codes, by appropriately changing the security game that is played in the second phase.

The above definition of usability is consistent with accepted definitions of symmetric key-based primitives based on security against adversaries that are allowed arbitrary uses of the primitive in a priori unknown settings. Our model adds the possibility that key generation is accomplished using a key exchange protocol instead of a non-interactive algorithm. The adversary is provided with auxiliary information obtained by interacting with this protocol.

2.1.3 Definitions

In this section, we define a trace property of protocols and show that this property implies computational secrecy and integrity. The computational secrecy properties include key indistinguishability and key usability for IND-CCA secure encryption. These results are established first for the simple case when secrets are protected by pre-shared "level-0" keys (Theorems 1-3), then generalized (Theorems 4-6) under the condition that each key is protected by predecessor keys in an acyclic graph. The proofs use standard cryptographic reductions.

Let s be a term of type nonce and $\mathcal{K}$ be a set of terms of type key. Intuitively, these terms will be used to set up a game in which an adversary associates s with a nonce value and $\mathcal{K}$ with keys that protect the nonce from the adversary. The adversary then tries to determine the value of the protected nonce. The protocol specification itself can be viewed as a blue-print which uses abstract names for the various terms being generated in a role. Each thread uses the blue-print of a specific role for the sequence of actions to be performed, but uses distinct names for the terms - so two different threads of the same role will call corresponding variables by different names. The terms s and $\mathcal{K}$ are names used by specific threads.

The adversary we consider knows the symbolic form of the protocol and determines the name of each term. An $(\mathsf{s}, \mathcal{K})$-adversary, $\mathcal{A}_{\mathsf{s},\mathcal{K}}$, is a protocol adversary that additionally chooses a thread to generate the nonce s, as well as chooses which keys in the execution to associate with the individual keys in $\mathcal{K}$. Since each thread consists of a principal executing a program for a role, adversary $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ may select a nonce from a trace by choosing a thread in that trace and the name used in that thread for a nonce, without knowing the bitstring value of the nonce, and similarly for the keys. For simplicity, we refer to the symbolic nonce selected by the adversary as s and the symbolic keys as $\mathsf{k} \in \mathcal{K}$. Given a trace $\langle e, \lambda\rangle$ and a choice of $\mathcal{K}$ by the adversary, let $\Lambda(\mathcal{K}) = \{\lambda(\mathsf{k}) \mid \mathsf{k} \in \mathcal{K}\}$ be the set of bitstring values of keys in $\mathcal{K}$. The bitstrings $\lambda(\mathsf{s})$ and $\lambda(\mathsf{k})$ for $\mathsf{k} \in \mathcal{K}$ are generally determined from the honest party randomness and are not a priori known to the adversary.

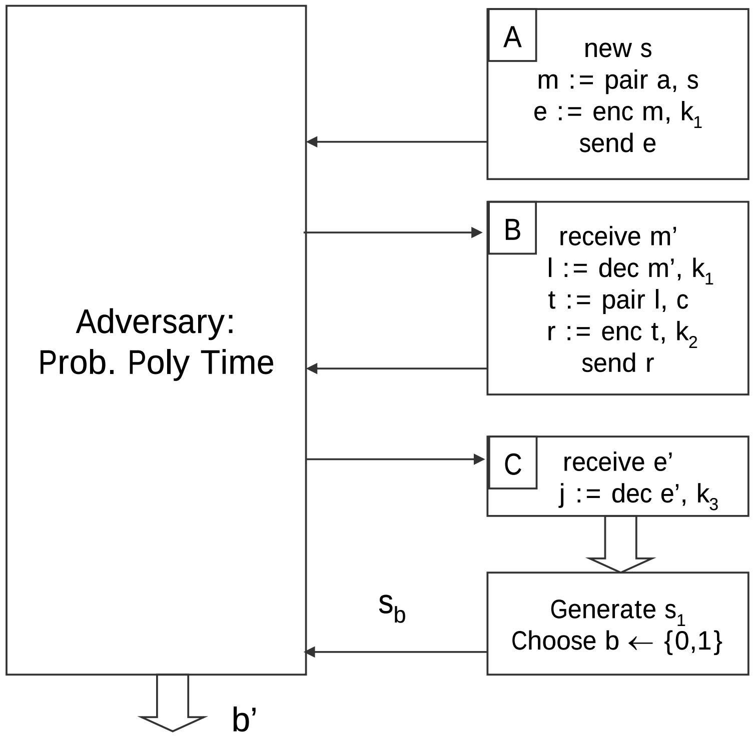
Intuitively, overlooking some details that we address in the definitions, a protocol with the following properties should guarantee the secrecy of s under the keys $\mathcal{K}$, in a way that is inductively verifiable:

- If the thread that generates the value of the nonce s sends this out on the network in any message, then the message must be structured such that s is encrypted with a key k with bitstring value $\lambda(\mathsf{k}) \in \Lambda(\mathcal{K})$.

- If any thread decrypts a message that was encrypted with a key k with $\lambda(\mathsf{k}) \in \Lambda(\mathcal{K})$ and sends any parts of this message out on the network, then those parts must be sent in a way that encrypts them with some key k' with $\lambda(\mathsf{k}') \in \Lambda(\mathcal{K})$.

In other words, s is protected by $\mathcal{K}$ if s is initially sent on the network in a form that is encrypted with one of the keys in $\mathcal{K}$, and any time a message encrypted with one of the keys (that might potentially contain s) is decrypted, any part of the resulting decryption must again be encrypted with a key in $\mathcal{K}$ before it is sent on the network. An example of a secretive protocol is given in Figure 2.1.

Definition 7 (Good Terms) *Let* s *be a term of type nonce and* $\mathcal{K}$ *be a set of terms of type key. Let* $\langle e, \lambda\rangle$ *be a trace generated by executing the protocol against an* $(\mathsf{s}, \mathcal{K})$*-adversary. An* $(\mathsf{s}, \mathcal{K})$-good *term for any thread in this trace is any term received by*



The adversary interacts with the protocol participants A, B, and C whose programs are written out. This is an $(\mathsf{s}, \mathcal{K})$-secretive protocol, where $\mathcal{K} = \{k_1, k_2, k_3\}$ because A sends out s encrypted with k_1 and B after decrypting m' with k_1 sends out the result l encrypted under k_2. At the end of the execution of the protocol, the indistinguishability test is carried out by generating a random number s_1 and a bit b and then sending s_b to the adversary (here the secret $s = s_0$). The adversary wins if the bit b' that she outputs is equal to b with non-negligibly greater probability than $1/2$.

Figure 2.1: A Secretive Protocol

that thread, a term of atomic type different from nonce or key, a nonce with value different from $\lambda(\mathsf{s})$*, or a term constructed in the following ways: pairing* $(\mathsf{s}, \mathcal{K})$*-good terms, unpairing an* $(\mathsf{s}, \mathcal{K})$*-good term, encrypting an* $(\mathsf{s}, \mathcal{K})$*-good term, encrypting any term with a key with value in* $\Lambda(\mathcal{K})$*, or decrypting good terms with keys with value not in* $\Lambda(\mathcal{K})$*.*

Definition 8 (Secretive Trace) *Let* s *be a term of type nonce and* $\mathcal{K}$ *be a set of terms of type key. A trace* $\langle e, \lambda \rangle$ *generated by executing the protocol against an* $(\mathsf{s}, \mathcal{K})$*-adversary is* $(\mathsf{s}, \mathcal{K})$-secretive *if every thread belonging to honest principals sends out only* $(\mathsf{s}, \mathcal{K})$-good *terms.*

Definition 9 (Secretive Protocol) *Let* $\mathcal{Q}$ *be a protocol and let* $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ *be an* $(\mathsf{s}, \mathcal{K})$*-adversary. Then* $\mathcal{Q}$ *is an* $(\mathsf{s}, \mathcal{K})$-secretive protocol *for* $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ *if, for all sufficiently large* η*, the probability that a trace* $t(\mathcal{A}_{\mathsf{s},\mathcal{K}}, \mathcal{Q}, \eta)$ *generated by the interaction of* $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ *with honest principals following roles of* $\mathcal{Q}$ *is* $(\mathsf{s}, \mathcal{K})$*-secretive is overwhelmingly close to 1, where this probability is taken over all adversary and protocol randomness. In formulas,*

$$(1 - Pr[t(\mathcal{A}_{\mathsf{s},\mathcal{K}}, \mathcal{Q}, \eta) \textit{ is } (\mathsf{s}, \mathcal{K})\textit{-secretive }]) \textit{ is a negligible function of } \eta$$

Recall that adversary $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ may choose arbitarily which nonce in the trace to designate as s and which keys to designate as elements of $\mathcal{K}$. In the Bellare et al. approach [16], the adversary fixes the threads about which a security property is to be established. Then s and $\mathsf{k} \in \mathcal{K}$ are specific terms occurring in those threads. In PCL [66], which provides a proof system for trace properties, a security property is usually in the form of post condition of a thread and this thread identifies the necessary term names. Although we use the results in this chapter to prove soundness of axioms and rules of PCL, we find it convenient here to leave the choice up to the adversary, since that leads to technical results that are applicable to any choice determined by the syntax of roles and formulas.

In proving properties of secretive protocols, we will be concerned with subset of traces that are secretive. Since the set of non-secretive traces is a negligible subset of all traces, by definition, any advantage the adversary obtains against the non-secretive traces will be cumulatively negligible. When clear from context, we will drop the subscripts $\mathsf{s}, \mathcal{K}$ from the adversary name.

A *level-0* key for a protocol execution is an encryption key which is only used as a key but never as part of a payload. We use multi-party security definitions due to Bellare, Boldyreva and Micali [13] applied to symmetric encryption schemes in the following theorems. In [13], IND-CCA and the multi-party IND-CCA game are shown to be asymptotically equivalent.

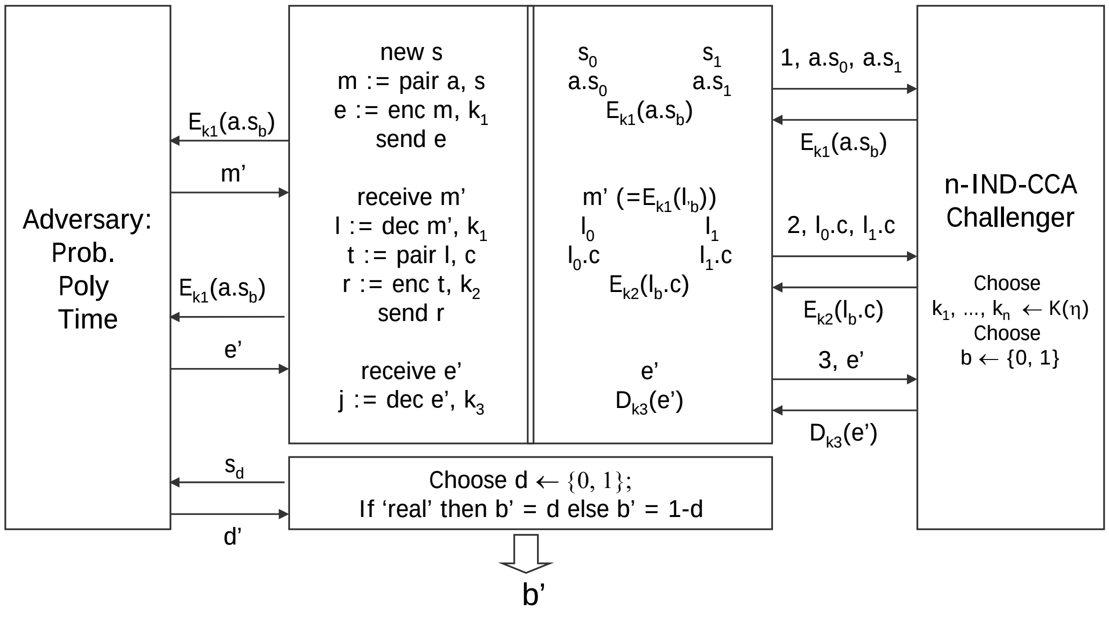
2.1.4 Bilateral Simulator

The general structure of the proofs of the secrecy theorems is by reduction of the appropriate protocol secrecy game to a multi-party IND-CCA game. That is, given protocol adversary $\mathcal{A}$, we construct an adversary $\mathcal{A}'$ against a multi-party IND-CCA challenger which provides $|\mathcal{K}|$-party Left-or-Right encryption oracles $\mathcal{E}_{k_i}(LR(\cdot,\cdot,b))$ parameterized by a challenge bit b and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}$ (Following [13], $LR(m_0, m_1, b)$ is a function which returns m_b).

The strategy of $\mathcal{A}'$ is to provide a simulation of the *secretive protocol* to $\mathcal{A}$ by using these oracles such that the capability of $\mathcal{A}$ to break the indistinguishability or key usability of the nonce can be leveraged in some way to guess the challenge bit b of the multi-party IND-CCA challenger. To this end, $\mathcal{A}'$ employs a *bilateral simulator* $\mathcal{S}$ which extracts two bit-strings s_0, s_1 from the randomness of $\mathcal{A}'$ as alternate values of the putative secret s and then simulates execution of the protocol to the protocol adversary $\mathcal{A}$ for both the values. The security parameter, the key generation algorithm, the symmetric encryption/decryption algorithms are fixed beforehand and are known to the simulator. The nonce length and key lengths are non constant positive polynomials in the security parameter. All keys not in $\mathcal{K}$ are generated by $\mathcal{S}$ as and when required. $\mathcal{A}$ can ask $\mathcal{S}$ for keys known to a dishonest party, but not keys known only to honest parties. Since $\mathcal{A}$ is polynomially bounded, $\mathcal{S}$ only has to generate polynomially bounded number of keys. All the randomness required by $\mathcal{S}$ in the various cryptographic operations, as well as the randomness required by $\mathcal{A}$, are supplied from the randomness of $\mathcal{A}'$. The IND-CCA challenger, however, uses independent randomness unknown to $\mathcal{A}'$. Both the randomness lengths are non-constant positive polynomials in the security parameter. All the security theorems in this chapter are over uniform probability distributions on both adversary and challenger randomness. The theorems are asymptotic in flavour in that they assert that for given polynomials specifying various parameter lengths and a given adversary, there is a large enough security parameter such that the adversary has negligible advantage in winning a game. Figure 2.2 illustrates the behavior of the bilateral simulator on one possible execution of the secretive protocol in Figure 2.1.

As with the execution of the actual protocol, $\mathcal{S}$ receives messages and scheduling information from $\mathcal{A}$ and acts according to the roles of the given protocol. $\mathcal{A}$ also specifies the symbolic names of the terms in the operations. The difference from a normal protocol execution is that in computing bitstring values of terms that involve s, $\mathcal{S}$ does so for both values of s. We will show that for secretive protocols the value of s that $\mathcal{A}$ sees is determined by the challenge bit b of the CCA challenger. The operational semantics of the bilateral simulator is formalized in Table 2.2. We explain the form of the definition using an example. The action $\mathsf{m} := \mathtt{pair}\ \mathsf{m}', \mathsf{m}''$ requires that the terms m and m' have already been evaluated in the protocol thread under execution. We also impose the syntactic requirement that each term variable is evaluated only once – this does not constrain the expressive power of the language as variables could be alpha renamed. The functions lv and rv map a term to its bit-string values intuitively corresponding to the values s_0 and s_1 of s respectively. The function *pair* is the actual computational implementation of pairing. The result of executing this action rule states that $lv(\mathsf{m})$ is evaluated by pairing the bit-strings $lv(\mathsf{m}')$ and $lv(\mathsf{m}'')$ and similarly for $rv(\mathsf{m})$. In simulating the protocol to the protocol adversary, the simulator executes each action of the currently scheduled thread following this definition.

We first argue informally here that the inductive structure of encryption and decryption allowed in a secretive protocol ensures that the simulator does not stop due to the operational semantics of a send action — a send action stops simulation when the $lv()$ and $rv()$ values of the the term to be sent are different. Lemma 1



The simulator simulates the execution of the protocol to the adversary, maintaining two copies of the secret `s`. The left half of the simulator shows the protocol actions it simulates and the right half shows the two values for the bitstrings ($lv(\mathsf{m})$ and $rv(\mathsf{m})$) following the operational semantics in Table 2.2. The `new s` action produces two random bitstrings s_0 and s_1. The pairing action then produces a left value $a.s_0$ and a right value $a.s_1$. The subsequent encryption action is simulated by making a call to the encryption oracle with the message pair $\langle a.s_0, a.s_1\rangle$; the index 1 indicates that the key is k_1. The enryption oracle outputs the encryption $E_{k_1}(a.s_b)$ using its internal random bit b. This encryption is then sent to the protocol adversary. The important point to note is that although there are two values of terms involving the secret, there is a *unique* value for messages that are sent to the adversary because terms containing the secret are always encrypted using the encryption oracle before transmission; this is guaranteed by the definition of secretive protocol. The rest of the simulation proceeds in a similar manner. At the end, a random bit d is generated and s_d is sent to the protocol adversary. If the protocol adversary says that s_d is the real secret, then the IND-CCA adversary outputs $b' = d$ else it outputs $b' = 1 - d$. Thus, if the protocol adversary wins the indistinguishability test with non-negligible advantage, the IND-CCA adversary wins that game with the same advantage.

Figure 2.2: The Bilateral Simulator

Assignment actions:

ACTION	$lv(\mathsf{m})$	$rv(\mathsf{m})$
m in the static list, $\mathsf{m} \notin \mathcal{K}$	value from initial input	equal to $lv(\mathsf{m})$
$\mathtt{receive}\ \mathsf{m}$;	receive from adversary	equal to $lv(\mathsf{m})$
$\mathtt{new}\ \mathsf{m}$; $\quad \mathsf{m} \neq \mathsf{s}$	generate nonce	equal to $lv(\mathsf{m})$
$\mathtt{new}\ \mathsf{m}$; $\quad \mathsf{m} = \mathsf{s}$	s_0	s_1
$\mathsf{m} := \mathtt{pair}\ \mathsf{m}', \mathsf{m}''$;	$pair(lv(\mathsf{m}'), lv(\mathsf{m}''))$	$pair(rv(\mathsf{m}'), rv(\mathsf{m}''))$
$\mathsf{m} := \mathtt{fst}\ \mathsf{m}'$;	$fst(lv(\mathsf{m}'))$	$fst(rv(\mathsf{m}'))$
$\mathsf{m} := \mathtt{snd}\ \mathsf{m}'$;	$snd(lv(\mathsf{m}'))$	$snd(rv(\mathsf{m}'))$
$\mathsf{m} := \mathtt{enc}\ \mathsf{m}', \mathsf{k}$; $\quad \mathsf{k} \in \mathcal{K}$ do $\begin{cases} qdb_k \leftarrow qdb_k \cup \{lv(\mathsf{m})\} \\ dec_k^0(lv(\mathsf{m})) \leftarrow lv(\mathsf{m}') \\ dec_k^1(lv(\mathsf{m})) \leftarrow rv(\mathsf{m}') \end{cases}$	$\mathcal{E}_k(LR(lv(\mathsf{m}'), rv(\mathsf{m}'), b))$	$\mathcal{E}_k(LR(lv(\mathsf{m}'), rv(\mathsf{m}'), b))$
$\mathsf{m} := \mathtt{dec}\ \mathsf{m}', \mathsf{k}$; $\quad \mathsf{k} \in \mathcal{K}$	$\begin{cases} \mathcal{D}_k(lv(\mathsf{m}')), \\ \qquad \text{if } lv(\mathsf{m}') \notin qdb_k \\ dec_k^0(lv(\mathsf{m}')), \\ \qquad \text{if } lv(\mathsf{m}') \in qdb_k \end{cases}$	$\begin{cases} \mathcal{D}_k(rv(\mathsf{m}')), \\ \qquad \text{if } rv(\mathsf{m}') \notin qdb_k \\ dec_k^0(rv(\mathsf{m}')), \\ \qquad \text{if } rv(\mathsf{m}') \in qdb_k \end{cases}$
$\mathsf{m} := \mathtt{enc}\ \mathsf{m}', \mathsf{n}$; $\quad \mathsf{n} \notin \mathcal{K}$, n tagged nonce (including s) or key	$enc(lv(\mathsf{m}'), lv(\mathsf{n}))$	$enc(rv(\mathsf{m}'), lv(\mathsf{n}))$
$\mathsf{m} := \mathtt{dec}\ \mathsf{m}', \mathsf{n}$; $\quad \mathsf{n} \notin \mathcal{K}$, n tagged nonce (including s) or key	$dec(lv(\mathsf{m}'), lv(\mathsf{n}))$	$dec(rv(\mathsf{m}'), lv(\mathsf{n}))$

Non-assignment actions:

ACTION	RESULT
$\mathtt{match}\ \mathsf{m}\ \mathtt{as}\ \mathsf{m}'$;	$\begin{cases} \text{continue} & \text{if } (lv(\mathsf{m}) = lv(\mathsf{m}')) \wedge (rv(\mathsf{m}) = rv(\mathsf{m}')) \\ \text{stop thread} & \text{if } (lv(\mathsf{m}) \neq lv(\mathsf{m}')) \wedge (rv(\mathsf{m}) \neq rv(\mathsf{m}')) \\ \text{stop simulation} & \text{otherwise} \end{cases}$
$\mathtt{send}\ \mathsf{m}$;	$\begin{cases} \text{send } lv(\mathsf{m}) \text{ to adversary} & \text{if } lv(\mathsf{m}) = rv(\mathsf{m}) \\ \text{stop simulation} & \text{otherwise} \end{cases}$

Table 2.2: Operational Semantics of the Simulator with Parameters $\mathsf{s}, \mathcal{K}$

states the general case more formally. Suppose m is a term explicitly constructed from s in the thread that generates s. As $\mathcal{S}$ is simulating an $(\mathsf{s}, \mathcal{K})$-*secretive protocol*, this term is to be encrypted with a key k in $\mathcal{K}$ to construct a message to be sent out to $\mathcal{A}$. So, $\mathcal{S}$ asks the encryption oracle of the $|\mathcal{K}|$-IND-CCA challenger to encrypt $(lv(\mathsf{m}), rv(\mathsf{m}))$ with k. In addition, this pair of bitstrings is recorded and the result of the query is logged in a set which we denote as qdb_k—for each key k we have a distinct set qdb_k. If a message construction involves decryption with a key in $\mathcal{K}$, $\mathcal{S}$ first checks whether the term to be decrypted was produced by an encryption oracle by accessing the log qdb_k—if not, then the decryption oracle is invoked; if yes, then $\mathcal{S}$ uses the corresponding encryption query as the decryption. In the second case the encryption query must have been of the form (m_0, m_1). Following the definition of *secretive protocol*, terms constructed from this decryption will be re-encrypted with a key in $\mathcal{K}$ before sending out. Thus we note here that all such replies will be consistent to $\mathcal{A}$ with respect to any choice of b.

The situation becomes trickier when encryption or decryption of a term is required with s as the key. In this case $\mathcal{S}$ encrypts or decrypts with s_0. We therefore always have $lv(\mathsf{m}) = rv(\mathsf{m})$ for any message m being sent out. Hence, the simulator will not get stuck due to a send action. The computational evaluation of an encryption using a nonce as a key involves generation of a key from the nonce using a deterministic *keygen* function. The *keygen* function is assumed to map a uniform distribution over the nonce space to a distribution computationally indistinguishable from the key space required by the symmetric encryption scheme.

One subtle issue arises when we consider term deconstructors such as unpairings and decryptions, and pattern matching actions. The bilateral simulation for term constructions like pairing and encryption is fairly straightforward. However, to have a consistent simulation we need to ensure that the success of the decontruction and pattern matching actions are independent of the challenge bit b, *i.e.* if the term for $b = 0$ can be unpaired or decrypted then the corresponding operation also succeeds for the term for $b = 1$; if there is a match for $b = 0$ then there should also be a match for $b = 1$.

For this technical reason, we assume that honest parties conform to certain type conventions. These restrictions may be imposed by prefixing the values of each type (nonces, ids, constant strings, pairs, encryptions with key k, etc.) with a tag such as 'constant' or 'encrypted with key-id $k - id$' that are respected by honest parties executing protocol roles. The adversary may freely modify or spoof tags or produce arbitrary untagged bitrings. It turns out that the type information carried by terms ensures deconstruction and matching consistency in an overwhelming number of traces. This is stated in Lemmas 2 and 3. The proofs proceed by induction over the operational semantics in Table 2.2.

Lemma 1 *If an honest principal in a trace $\langle e, \lambda \rangle$ constructs an $(\mathsf{s}, \mathcal{K})$-good term m, then any bilateral simulator with parameters $s, \mathcal{K}$, executing symbolic actions e produces identical bitstring values for m on both sides of the simulation,* i.e., *we will have $lv(\mathsf{m}) = rv(\mathsf{m})$.*

Proof. The proof is by induction on the construction of good terms. The base cases for received terms, terms of atomic type different from nonce or key simply follow from the operational semantics of the simulator. For encryption of a good term and decryption with a key not in $\mathcal{K}$, we use the fact that only the $lv()$ of the key is used in the operational semantics for encryption and decryption, hence the result also has equal $lv()$ and $rv()$ values. The case for encryption of any term with a key in $\mathcal{K}$ follows as the operational semantics for this case produces the same value for $lv()$ and $rv()$ in the result. $\square$

Definition 10 ($\cong$) *For term variables m, m', we write $\mathsf{m} \cong \mathsf{m}'$ iff $lv(\mathsf{m}) = lv(\mathsf{m}') \wedge rv(\mathsf{m}) = rv(\mathsf{m}')$.*

Lemma 2 (Consistent Deconstructions) *If the bitstring value of the term variable m is a pair on one side of a bilateral simulation then it is a pair on the other side also; similarly for encryption. Formally,*

- *If $lv(\mathsf{m}) = pair(l_0, l_1)$ for some l_0, l_1, then $rv(\mathsf{m}) = pair(r_0, r_1)$ for some r_0, r_1 and vice versa.*

- *If $lv(\mathsf{m})$ is equal to an encryption $enc(l, lv(\mathsf{k}))$ for some l, then $rv(\mathsf{m})$ is also equal to an encryption $enc(r, lv(\mathsf{k}))$ for some r and vice versa (It is implicit that the encryption is randomized).*

Proof. The proof is by simultaneous induction on the following stronger propositions:

- If $lv(\mathsf{m}) = pair(l_0, l_1)$ for some l_0, l_1, then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or there exists m' such that $\mathsf{m} \cong \mathsf{m}'$ and m' is derived by a `pair` action.

- If $lv(\mathsf{m}) = enc(l, lv(\mathsf{k}))$ for some l, then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or, $\mathsf{k} \notin \mathcal{K}$ and there exists m', k' such that $\mathsf{m} \cong \mathsf{m}', \mathsf{k} \cong \mathsf{k}'$ and m' is derived by an `enc` $\cdot, \mathsf{k}'$ action.

- If $lv(\mathsf{m})$ is tagged to be of type nonce then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or, $lv(\mathsf{m}) = s_0 \wedge rv(\mathsf{m}) = s_1$.

- In all other cases, $lv(\mathsf{m}) = rv(\mathsf{m})$

We do the induction as follows on the operational semantics defined in Table 2.2:

Case: (m in the static list, $\mathsf{m} \notin \mathcal{K}$), (`receive` m;), (`new` m; $\quad \mathsf{m} \neq \mathsf{s}$) – in all these cases, $lv(\mathsf{m}) = rv(\mathsf{m})$, so we are done.

Case: `new` m; $\quad \mathsf{m} = \mathsf{s}$ – this satisfies $lv(\mathsf{m}) = s_0 \wedge rv(\mathsf{m}) = s_1$.

Case: $\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$; – resultant value is a pair of two component values and satisfies the proposition that it has been derived by a `pair` action.

Case: $\mathsf{m} :=$ `enc` m', n; $\quad \mathsf{n} \notin \mathcal{K}$ – resultant value is an encryption using the key $lv(\mathsf{n})$ values and satisfies the proposition that it has been derived by a `enc` action.

Case: $\mathsf{m} :=$ `enc` m', n; $\quad \mathsf{n} \in \mathcal{K}$ – in all this case, $lv(\mathsf{m}) = rv(\mathsf{m})$

Case: $\mathsf{m} :=$ `fst` m'; Now, $lv(\mathsf{m}') = pair(lv(\mathsf{m}), l)$ for some l. Therefore, by IH, either $lv(\mathsf{m}') = rv(\mathsf{m}')$ or m' was derived by a `pair` action.

If $lv(\mathsf{m}') = rv(\mathsf{m}')$, then $lv(\mathsf{m}) = fst(lv(\mathsf{m}')) = fst(rv(\mathsf{m}')) = rv(\mathsf{m})$ and we are done.

If $\mathsf{m}'' \cong \mathsf{m}'$ was derived by a `pair` action, say: $\mathsf{m}'' := \texttt{pair}\ \mu, \mu'$; Therefore,

$$lv(\mathsf{m}'') = lv(\mathsf{m}') \Rightarrow fst(lv(\mathsf{m}'')) = fst(lv(\mathsf{m}')) \Rightarrow lv(\mu) = lv(\mathsf{m})$$

$$rv(\mathsf{m}'') = rv(\mathsf{m}') \Rightarrow fst(rv(\mathsf{m}'')) = fst(rv(\mathsf{m}')) \Rightarrow rv(\mu) = rv(\mathsf{m})$$

Therefore $\mu \cong \mathsf{m}$ and we are through by IH. The induction over rule `SND` is similar.

Case: $\mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{k}$; Now, $lv(\mathsf{m}') = enc(lv(\mathsf{m}), lv(\mathsf{k}))$. Therefore, by IH, either $lv(\mathsf{m}') = rv(\mathsf{m}')$ or, $\mathsf{k} \notin \mathcal{K}$ and some m'' was derived by an $\texttt{enc}\ \cdot, \mathsf{k}''$ action, with $\mathsf{m}'' \cong \mathsf{m}', \mathsf{k}'' \cong \mathsf{k}'$.

If $lv(\mathsf{m}') = rv(\mathsf{m}')$ and their value is $\in qdb_k$ then rule for the case ($\mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{k}$; $\mathsf{k} \in \mathcal{K}$) must have been invoked for an $\mathsf{m}'' \cong \mathsf{m}'$ at some point before, when the value was entered in the qdb_k - i.e. we had $\mathsf{m}'' := \texttt{enc}\ \mu, \mathsf{k}$; $\mathsf{k} \in \mathcal{K}$. Therefore $lv(\mu) = dec^0_k(lv(\mathsf{m}'')) = dec^0_k(lv(\mathsf{m}')) = lv(\mathsf{m})$, and $rv(\mu) = dec^1_k(rv(\mathsf{m}'')) = dec^1_k(rv(\mathsf{m}')) = rv(\mathsf{m})$. Hence, $\mu \cong \mathsf{m}$ and the induction follows by IH.

If $lv(\mathsf{m}') = rv(\mathsf{m}')$ and their value is $\notin qdb_k$, then $enc(lv(\mathsf{m}), lv(k)) = enc(rv(\mathsf{m}), lv(k))$. Therefore $lv(\mathsf{m}) = rv(\mathsf{m})$.

If $lv(\mathsf{m}') \neq rv(\mathsf{m}')$, then $\mathsf{k} \notin \mathcal{K}$ and some m'' was derived by an $\texttt{enc}\ \cdot, \mathsf{k}''$ action, with $\mathsf{m}'' \cong \mathsf{m}', \mathsf{k}'' \cong \mathsf{k}'$. Since $lv(\mathsf{m}') \neq rv(\mathsf{m}')$, this derivation must have been by the rule for the action ($\mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{k}$; $\mathsf{k} \in \mathcal{K}$), say: $\mathsf{m}'' := \texttt{enc}\ \mu, \mathsf{k}''$. Therefore,

$$lv(\mathsf{m}'') = lv(\mathsf{m}') \Rightarrow enc(lv(\mu), lv(\mathsf{k}'')) = enc(lv(\mathsf{m}), lv(\mathsf{k})) \Rightarrow lv(\mu) = lv(\mathsf{m})$$

$$rv(\mathsf{m}'') = rv(\mathsf{m}') \Rightarrow enc(rv(\mu), lv(\mathsf{k}'')) = enc(rv(\mathsf{m}), lv(\mathsf{k})) \Rightarrow rv(\mu) = rv(\mathsf{m})$$

Therefore $\mu \cong \mathsf{m}$ and we are through by IH.

Lemma 3 (Consistent Matchings) *If the term variables* m *and* m' *are computationally evaluated by a bilateral simulator, and the action* ***`match`*** m ***`as`*** m' *is executed, then with overwhelming probability, the match succeeds on the left side iff it succeeds* *on the right.*

Proof. Suppose the action $\texttt{match}\ \mathsf{m}\ \texttt{as}\ \mathsf{m}'$ is executed and the match succeeds exactly on one side. Therefore, for at least one of m, m'—say m'—it must be that $lv(\mathsf{m}') \neq rv(\mathsf{m}')$. Therefore, m' must have been derived by either pairing or, an encryption with a key k such that $\mathsf{k} \notin \mathcal{K}$, or, $\mathsf{m}' \cong \mathsf{s}$.

- Suppose $\mathsf{m}'', \mathsf{m}'' \cong \mathsf{m}'$, was derived by a $\texttt{pair}$ action – $\mathsf{m}'' = \texttt{pair}\ \mu, \mu'$. It must be that for one of μ, μ', say μ, $lv(\mu) \neq rv(\mu)$.
- Suppose $\mathsf{m}'', \mathsf{m}'' \cong \mathsf{m}'$, was derived by a $\texttt{enc}$ action – $\mathsf{k} \notin \mathcal{K}, \mathsf{m}'' = \texttt{enc}\ \mu, \mathsf{k}$. It must be that $lv(\mu) \neq rv(\mu)$.

Now the argument about m' can be reapplied to μ since $lv(\mu) \neq rv(\mu)$. It therefore follows that as we iteratively look at the derivations, we will eventually reach a symbol n, such that $\mathsf{n} \cong \mathsf{s}$. Intuitively what this means is that it is possible to obtain s from m' through a series of unpairings and decryptions using keys that honest principals have used. We will be guided by this intuition to build an adversary against a $|\mathcal{K}|$-IND-CCA challenger.

The $|\mathcal{K}|$-IND-CCA adversary we build consists of a modified simulator $\mathcal{S}'$ interacting with a protocol adversary $\mathcal{A}'$ which can instruct $\mathcal{S}'$ to undertake individual actions, represented symbollically. As compared to $\mathcal{S}$, $\mathcal{S}'$ follows the same operational semantics on individual symbolic actions from the given protocol syntax but instead of conforming to a specific protocol role, these actions are dictated by the adversary. In addition $\mathcal{S}'$ outputs status of actions such as 'match succeeded or failed on left, right or both sides', 'unpairing succeeded/failed' and 'decryption succeeded/failed', 'send failed as $lv() \neq rv()$'.

The algorithm $\mathcal{A}'$ first uses $\mathcal{S}'$ to simulate the protocol execution to the protocol adversary $\mathcal{A}$. Whenever the action $\texttt{match}\ \mathsf{m}\ \texttt{as}\ \mathsf{m}'$ is executed and the match succeeds exactly on one side, the simulator $\mathcal{S}'$ tells $\mathcal{A}'$ so. At this point, $\mathcal{A}'$ executes the function $\texttt{parse-get-s}$ (Table 2.3) on the symbols m and m'. We will show that for at least one of m and m', $\texttt{parse-get-s}()$ will output a (b', s') such that $s' = s_{b'}$. Also with overwhelming probability $b' = b$ will hold, where b is the $|\mathcal{K}|$-IND-CCA challenger bit.

```
Algorithm parse-get-s(m)
  if ($\mathcal{S}'$ : m' := fst m; ) succeeds then        [m' is a new symbol]
    $r \leftarrow$ parse-get-s(m')
    if ($r \neq \bot$) return $r$
    else $\mathcal{S}'$ : m'' := snd m;        [m'' is a new symbol]
    return parse-get-s(m'')
  else do
    for all k evaluated so far  and k $\notin \mathcal{K}$
      if($\mathcal{S}'$ : m' := dec m, k; ) succeeds then        [m' is a new symbol]
        return parse-get-s(m')
  else do
    if(match m as s) succeeds on exactly one side
      $m \leftarrow$ result($\mathcal{S}'$ : send m; )
      if match succeeded on left side then $b \leftarrow 0$ else $b \leftarrow 1$
      return $(b, m)$
    else return $\bot$
  else return $\bot$
```

Table 2.3: Algorithm parse-get-s

Suppose the matching succeeds on the left side, i.e., $lv(\mathsf{m}) = lv(\mathsf{m}')$ but $rv(\mathsf{m}) \neq rv(\mathsf{m}')$. Since $lv(\mathsf{m}) = lv(\mathsf{m}')$, note that m and m' will have an identical parsing sequence. Since also $lv(\mathsf{m}') \neq rv(\mathsf{m}')$, the parsing should reach, at least once, an n' with $\mathsf{n}' \cong \mathsf{s}$. The parse tree of m should also have a corresponding n with $lv(\mathsf{n}) = s_0$. Since $rv(\mathsf{m}) \neq rv(\mathsf{m}')$, for at least one of these leaves n we should have $rv(\mathsf{n}) \neq s_1$, but in this case we can only have $rv(\mathsf{n}) = s_0 = lv(\mathsf{n})$. Therefore `send n` action is allowed for $\mathcal{S}'$. Therefore `parse-get-s`(m) succeeds and outputs $(0, s_0)$.

Case 1: matching succeeds only in the left. We argue in this case that $b = 0$ with overwhelming probability. Suppose on the contrary, $b = 1$. Then the protocol simulation from $\mathcal{A}'$'s perspective used s_0 only as a key to an IND-CCA secure encryption scheme, if used at all. Since s_0 was chosen randomly, the probability of an adversary coming up with s_0 is negligible. Hence $b = 1$ only with negligible probability.

Case 2: matching succeeds only in the right. We argue in this case that $b = 1$ with overwhelming probability. Suppose on the contrary, $b = 0$. Then the protocol simulation from $\mathcal{A}'$'s perspective never used s_1. Since s_1 was chosen randomly, the probability of an adversary to come up with s_1 is negligible. Hence $b = 0$ only with negligible probability. $\square$

2.1.5 Computational Security

We recall some of the earlier discussion here for context. An $(\mathsf{s}, \mathcal{K})$-adversary, $\mathcal{A}_{\mathsf{s},\mathcal{K}}$, is a protocol adversary that additionally chooses a thread to generate the nonce s, as well as chooses which keys in the execution to associate with the individual keys in $\mathcal{K}$. Since each thread consists of a principal executing a program for a role, adversary $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ may select a nonce from a trace by choosing a thread in that trace and the name used in that thread for a nonce, without knowing the bitstring value of the nonce, and similarly for the keys. In proving properties of secretive protocols, we will be concerned with subset of traces that are secretive. Since the set of non-secretive traces is a negligible subset of all traces, by definition, any advantage the adversary obtains against the non-secretive traces will be cumulatively negligible. A *level-0* key for a protocol execution is an encryption key which is only used as a key but never as part of a payload.

Theorem 1 (CCA security - No keying - level 0) *Assume that a probabilistic poly-time* $(\mathsf{s}, \mathcal{K})$*-adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s}, \mathcal{K})$*-secretive protocol with* $\mathcal{K}$ *consisting of level-0 keys only. Also assume that in every trace* $\langle e, \lambda\rangle$*, a term of value* $\lambda(\mathsf{s})$ *is never used as a key by the honest principals. The adversary has negligible advantage at distinguishing the value of* s *from random, over the set of all traces* $\langle e, \lambda\rangle$*, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies key indistinguishability for* s *against* $\mathcal{A}$*.*

Proof. We will show that if $\mathcal{A}$ has non-negligible advantage at distinguishing the value of s from random, after the interaction, then we can construct a $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ with non-negligible advantage against the encryption scheme.

Adversary $\mathcal{A}_1$ employs a bilateral simulator $\mathcal{S}$ which randomly chooses two bit-strings s_0, s_1 as alternate values of the putative secret s and then simulates execution of the protocol to the protocol adversary $\mathcal{A}$ for both the values. Since the protocol is $(\mathsf{s}, \mathcal{K})$-secretive with respect to $\mathcal{A}$, terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ values by Lemma 1. By Lemmas 2 and 3, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation. Since s is never used as a key, we therefore observe that the simulated protocol behaviour will be consistent to $\mathcal{A}$ with respect to any choice of b.

In the second phase, $\mathcal{A}_1$ chooses a bit d' and sends $s_{d'}$ to $\mathcal{A}$. If $\mathcal{A}$ replies that this is the actual nonce used, then $\mathcal{A}_1$ finishes by outputting $d = d'$, otherwise it outputs $d = \bar{d}'$ and finishes. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \tag{2.1}$$

Since $\mathcal{A}$ has a non-negligible advantage at distinguishing s from random, the quantity on the RHS must be non-negligible. Therefore the advantage in the LHS must be non-negligible and hence we are done. □

If a protocol is an $(\mathsf{s}, \mathcal{K})$-*secretive protocol* where $\mathcal{K}$ is a set of level 0 keys, then we will call s a level-1 key for the protocol, protected by $\mathcal{K}$. We provide below an example of a protocol with level-1 keys and then state a theorem establishing key usability for level-1 keys.

Example. Consider the following two party protocol **KeyEx**:

Initiator A's program for a session with B:

```
new n;
m := enc n, $k_{A,B}$;
send m;

receive u;
v := dec u, n;
match v as $B$;
```

Responder B's program for a session with A:

```
receive p;
r := dec p, $k_{A,B}$;
t := enc $B$, r;
send t;
```

Consider an $(\mathsf{s}, \{\mathsf{k}_{P,Q}\})$-adversary $\mathcal{A}$ which picks a thread of principal P to execute the initiator role with responder Q and names the nonce generated by P as s and the shared key between P and Q as $\mathsf{k}_{P,Q}$. We claim that **KeyEx** is an $(\mathsf{s}, \{\mathsf{k}_{P,Q}\})$ secretive protocol for $\mathcal{A}$. The informal proof is as follows: The thread P encrypts s by the key $\mathsf{k}_{P,Q}$. Only a thread of P or Q can decrypt this message. As can be seen from the protocol structure, any thread that decrypts with the key $\mathsf{k}_{P,Q}$, does not send out part of the result of the decryption as a payload in another message. Note that Q uses the key s to encrypt and produce the term $ENC[\mathsf{s}](Q)$ to be sent out. Theorem 2 establishes that s satisfies key usability against $\mathcal{A}$.

Theorem 2 (CCA security - Keying - level 1) *Assume that a probabilistic poly-time* $(\mathsf{s}, \mathcal{K})$*-adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s}, \mathcal{K})$*-secretive protocol with* $\mathcal{K}$ *consisting of level-0 keys only. Honest principals are allowed to use a term of value* $\lambda(\mathsf{s})$ *as a key in every trace* $\langle e, \lambda\rangle$*. The adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key* $\lambda(\mathsf{s})$*, over the set of all traces* $\langle e, \lambda\rangle$*, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies IND-CCA key usability for* s *against* $\mathcal{A}$*.*

Proof. We will show that if $\mathcal{A}$ has non-negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key $\lambda(\mathsf{s})$, after the interaction then we can construct either a $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ or an IND-CCA adversary $\mathcal{A}_2$ with non-negligible advantages against the encryption scheme.

We proceed as in the proof of Theorem 1 to construct the adversary $\mathcal{A}_1$. The situation becomes different when encryption or decryption of a term is required with a nonce or a key n which has different $lv()$ and $rv()$ values. As we saw in the proof of Lemma 2, this can only happen when $lv(\mathsf{n}) = s_0$ and $rv(\mathsf{n}) = s_1$. In this case $\mathcal{S}$ encrypts or decrypts with $lv(\mathsf{n})$, that is, s_0. Again, as in the proof of Theorem 1, terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ values by Lemma 1 and by Lemmas 2 and 3, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation.

In the second phase, $\mathcal{A}_1$ uniformly randomly chooses a bit b' and provides oracles $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b'))$ and $\mathcal{D}_{s_0}(\cdot)$ to $\mathcal{A}$ for an IND-CCA game. $\mathcal{A}$ finishes by outputting a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs $d = 0$ else outputs $d = 1$. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d=0|b=0] - Pr[d=0|b=1] \tag{2.2}$$

Observe that if $b = 0$ then s was consistently represented by s_0 in messages sent to $\mathcal{A}$. Hence, the first probability is precisely the probability of $\mathcal{A}$ winning an IND-CCA challenge with $\lambda(\mathsf{s})$ as the key after interacting with an $(\mathsf{s}, \mathcal{K})$-*secretive protocol.* We will now bound the second probability. We start by constructing a second adversary $\mathcal{A}_2$ which interacts with an alternate simulator $\mathcal{S}'$ (described in Table 2.4) which has all the keys in $\mathcal{K}$, randomly generates a nonce s_1 and has access to an encryption oracle $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b_1))$ and a decryption oracle $\mathcal{D}_{s_0}(\cdot)$. It has a similar behaviour towards $\mathcal{A}$ as $\mathcal{S}$ had except that when constructing terms with s, it uses s_1 but when required to encrypt or decrypt using s, it queries $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b_1))$ or $\mathcal{D}_{s_0}(\cdot)$. In the second phase, $\mathcal{A}_2$ uses the oracles $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b_1))$ and $\mathcal{D}_{s_0}(\cdot)$ to provide the IND-CCA challenger to $\mathcal{A}$. $\mathcal{A}$ finishes by outputting a bit d_1. $\mathcal{A}_2$ outputs d_1. We observe here that if $b = 1$ for the earlier LR oracle, it makes no difference to the algorithm $\mathcal{A}$ whether it is interacting with $\mathcal{A}_1$ or $\mathcal{A}_2$. Thus we have:

$$(1/2)\mathbf{Adv}_{IND-CCA,\mathcal{A}_2}(\eta) = Pr[d_1 = b_1] - 1/2 = Pr[d = 0|b = 1] - 1/2 \qquad (2.3)$$

By the Equations 2.2 and 2.3 we have:

$$Pr[d = 0|b = 0] - 1/2 = \mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{IND-CCA,\mathcal{A}_2}(\eta)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Now we state a theorem establishing the integrity of encryptions done with level-1 keys. The security definition INT-CTXT for ciphertext integrity is due to [14] and also referred to as *existential unforgeability* of ciphertexts in [51].

Theorem 3 (CTXT integrity - level 1) *Assume that a probabilistic poly-time* (s, $\mathcal{K}$)*-adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s},\mathcal{K})$*-secretive protocol with* $\mathcal{K}$ *consisting of level-0 keys only. In any trace* $\langle e,\lambda\rangle$*, if an honest principal decrypts a ciphertext with a key of value* $\lambda(\mathsf{s})$ *successfully, then with overwhelming probability, over the set of all traces* $\langle e,\lambda\rangle$*, the ciphertext was produced by an honest principal by encryption with a key of value* $\lambda(\mathsf{s})$ *if the encryption scheme is IND-CCA and INT-CTXT secure.*

Proof. Suppose during the protocol run, an honest party decrypts a ciphertext with a key of value $\lambda(\mathsf{s})$ successfully which was not produced by an honest party by encryption with $\lambda(\mathsf{s})$. We build a $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ against set of keys $\mathcal{K}$ in the lines of the proof of Theorem 2. However, this new $\mathcal{A}_1$ computes d in a different way. Recall that $\mathcal{S}$ uses s_0 when it intends to encrypt or decrypt using s. In the

Assignment actions:

ACTION	$v(\mathsf{m})$
m in the static list, $\mathsf{m} \notin \mathcal{K}$	value from initial input
`receive m;`	receive from adversary
`new m;` $\quad \mathsf{m} \neq \mathsf{s}$	generate nonce
`new m;` $\quad \mathsf{m} = \mathsf{s}$	s_1
$\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$;	$pair(v(\mathsf{m}'), v(\mathsf{m}''))$
$\mathsf{m} :=$ `fst` m';	$fst(v(\mathsf{m}'))$
$\mathsf{m} :=$ `snd` m';	$snd(v(\mathsf{m}'))$
$\mathsf{m} :=$ `enc` m', k; $\quad v(\mathsf{n}) = s_1$ do $\begin{cases} qdb \leftarrow qdb \cup \{v(\mathsf{m})\} \\ dec_{s_0}(v(\mathsf{m})) \leftarrow v(\mathsf{m}') \end{cases}$	$\mathcal{E}_{s_0}(LR(v(\mathsf{m}'), v(\mathsf{m}'), b))$
$\mathsf{m} :=$ `dec` m', k; $\quad v(\mathsf{n}) = s_1$	$\begin{cases} \mathcal{D}_{s_0}(v(\mathsf{m}')), \\ \quad \text{if } v(\mathsf{m}') \notin qdb \\ dec_{s_0}(v(\mathsf{m}')), \\ \quad \text{if } v(\mathsf{m}') \in qdb \end{cases}$
$\mathsf{m} :=$ `enc` m', k; $\quad v(\mathsf{k}) \neq s_1$	$enc(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} :=$ `dec` m', k; $\quad v(\mathsf{k}) \neq s_1$, n tagged nonce or key	$dec(v(\mathsf{m}'), v(\mathsf{k}))$

Non-assignment actions:

ACTION	RESULT
`match m as m'`;	$\begin{cases} \text{continue} & \text{if } v(\mathsf{m}) = v(\mathsf{m}') \\ \text{stop thread} & \text{otherwise} \end{cases}$
`send m;`	$\{$send $v(\mathsf{m})$ to adversary

Table 2.4: Operational Semantics of the Alternate Simulator with Parameters $\mathsf{s}, \mathcal{K}$ in the case $b = 1$

course of interaction with $\mathcal{A}_1$, if $\mathcal{S}$ succeeds in decrypting a ciphertext with key s_0 which was not produced at a previous stage by $\mathcal{S}$ by encryption with s_0, $\mathcal{A}_1$ outputs $d = 0$. Otherwise, it outputs $d = 1$. The simulator does not get stuck due to any send, deconstruction or matching action as we have seen in the previous two proofs. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \tag{2.4}$$

Now, $Pr[d = 0|b = 0]$ is the probability of $\mathcal{S}$ succeeding in decrypting a ciphertext with a key of value $\lambda(\mathsf{s})$ which was not obtained through encryption by $\mathcal{S}$. $Pr[d = 0|b = 1]$ is the probability of $\mathcal{A}_1$ succeeding in decrypting a ciphertext with level-0 key s_0 (as in this case s_0 was only used as a key). Therefore, using a similar idea as proof of Theorem 2 we can build an INT-CTXT adversary $\mathcal{A}_2$ against s_0. Therefore,

$$Pr[d = 0|b = 0] = \mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) + \mathbf{Adv}_{INT-CTXT,\mathcal{A}_2}(\eta)$$

As the encryption scheme is both IND-CCA and INT-CTXT secure, both the probabilities on the RHS must be negligible and hence the theorem. $\square$

We now extend Theorems 1–3 to directed key hierarchies. This extension is motivated by the fact that many key distribution protocols (e.g. Kerberos) have key hierarchies with keys protected by lower level keys in the hierarchy. We again mandate that the nonces and keys in the set $\mathcal{K}$ are indicated by the adversary while executing the threads.

Recall that for a finite directed graph, the immediate predecessors of a node is the set of nodes that have edges to it.

Definition 11 (Key Graph) *Let $\mathcal{K}$ be a set of terms of type nonce or key in a trace $\langle e, \lambda \rangle$. A directed graph Γ with elements of $\mathcal{K}$ as vertices is a* key graph *for $\mathcal{K}$ for the trace if the following holds: for every node k and the set of its immediate predecessors $\mathcal{K}'$, the trace is $(\mathsf{k}, \mathcal{K}')$-secretive. If T is a set of traces, then Γ is a key graph for $\mathcal{K}$ for T if it is a key graph for $\mathcal{K}$ for each trace in T.*

Recall that $\mathcal{K}$ is a set of symbolic names. When $\mathcal{K}$ only has level-0 keys, the names are globally agreed upon. For example, $\mathsf{sk}_{Alice,Bob}$ may be the name of the symmetric key shared by Alice and Bob. The adversary just knows how to represent the key symbolically, but he may not know the bitstring value of it. When $\mathcal{K}$ has nonces as well, the name of a specific nonce n may not be the same across different threads. The adversary decides which thread generates n - the generating thread will call the nonce by the name n; same for the putative secret nonce s. This decision is taken during the run. The nonce generating threads can all be distinct.

In a directed acyclic graph, a *root* is a node that has no predecessor. The *level* of a node in the graph is its maximum distance from a root, over all roots from which it is reachable.

Definition 12 (Key Level) *Let $\mathcal{K}$ be a set of terms of type nonce or key in a set of traces T. Let Γ be a directed acyclic key graph for $\mathcal{K}$ for T. The* level of a key *is its level in graph Γ.*

Definition 13 (Key Basis) *Let $\mathcal{K}$ be a set of terms of type nonce or key in a set of traces T. Let Γ be a directed acyclic key graph for $\mathcal{K}$ for T. We define its* basis, *$\mathcal{B}(\Gamma)$, to be the set of all keys at the root, i.e., $\mathcal{B}(\Gamma)$ is the set of level 0 keys in $\mathcal{K}$.*

Theorem 4 (CCA security - No Keying) *Assume that a probabilistic poly-time $(\mathsf{s}, \mathcal{K})$-adversary $\mathcal{A}$ interacts with an $(\mathsf{s}, \mathcal{K})$-secretive protocol such that there is a key graph Γ for $\mathcal{K}$ which is a DAG. Also assume that a key of value $\lambda(\mathsf{s})$ is never used as a key by the honest principals in every trace $\langle e, \lambda\rangle$. The adversary has negligible advantage at distinguishing $\lambda(\mathsf{s})$ from random, over the set of all traces $\langle e, \lambda\rangle$, after the interaction, if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies key indistinguishability for s against $\mathcal{A}$.*

Proof. We will prove this by induction over the maximum level of Γ. If $\mathcal{K}$ consists only of level 0 keys then the result follows from Theorem 1. Suppose the maximum level in Γ is $(n + 1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be $\mathcal{B}(\Gamma)$.

We will show that if $\mathcal{A}$ has non-negligible advantage at distinguishing $\lambda(\mathsf{s})$ from a random bitstring of the same length, after the interaction, then we can construct either a $|\mathcal{K}'|$-IND-CCA adversary $\mathcal{A}_1$ to the encryption scheme or contradict the induction hypothesis.

We will construct an adversary $\mathcal{A}_1$ which has access to a modified bilateral simulator $\mathcal{S}$ (described in Table 2.5) which, in turn, has access to multi-party LR encryption oracles $\mathcal{E}_{k_i}(LR(\cdot, \cdot, b))$ and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}'$ parameterized by a bit b chosen uniformly randomly. For keys s^i of level > 0, $\mathcal{S}$ chooses random values s_0^i, s_1^i and for s, $\mathcal{S}$ chooses random values s_0, s_1. Intuitively, $\mathcal{S}$ constructs messages to be sent to $\mathcal{A}$ as follows:

- to encrypt the term $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with $\mathsf{k}_i \in \mathcal{K}'$, use response to oracle query $\mathcal{E}_{k_i}(LR(f(s_0, s_0^1, s_0^2, ...), f(s_1, s_1^1, s_1^2, ...), b))$.
- to encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with s^i, use $\mathcal{E}_{s_0^i}(f(s_0, s_0^1, s_0^2, ...))$.

Decryption operations are served analogously. The same idea of the proof for Lemma 1 still applies to the operational semantics in this setting, hence terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ as the protocol is $(\mathsf{s}, \mathcal{K})$-secretive with respect to $\mathcal{A}$. The proofs of Lemmas 2 and 3 are similar to the ones with semantics for level 0 keys in $\mathcal{K}$. Therefore, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation.

In the second phase, $\mathcal{A}_1$ chooses a bit d' and sends $s_{d'}$ to $\mathcal{A}$. If $\mathcal{A}$ replies that this is the actual nonce used, then $\mathcal{A}_1$ finishes by outputting $d = d'$, otherwise it outputs $d = \bar{d'}$ and finishes. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}'|$-IND-CCA challenger is:

$$\begin{aligned}\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) &= Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \\ &= (Pr[d = 0|b = 0] - 1/2) \\ &\quad + (Pr[d = 1|b = 1] - 1/2) \end{aligned} \tag{2.5}$$

The first probability in the RHS is precisely the probability of $\mathcal{A}$ breaking the indistinguishability of s_0 or equivalently of s. In the case when $b = 1$, the terms were constructed in the following manner:

- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with $\mathsf{k}_i \in \mathcal{K}'$: $\mathcal{E}_{k_i}(f(s_1, s_1^1, s_1^2, ...))$.

ACTION	$lv(\mathsf{m})$	$rv(\mathsf{m})$
$\mathtt{new\ m};\quad \mathsf{m} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	generate nonce	equal to $lv(\mathsf{m})$
$\mathtt{new\ m};\quad \mathsf{m} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	generate nonce	generate another nonce
$\mathsf{m} := \mathtt{enc}\ \mathsf{m}', \mathsf{k};\quad \mathsf{k} \in \mathcal{B}(\mathcal{K})$ do $\begin{cases} qdb_k \leftarrow qdb_k \cup \{lv(\mathsf{m})\} \\ dec_k^0(lv(\mathsf{m})) \leftarrow lv(\mathsf{m}') \\ dec_k^1(lv(\mathsf{m})) \leftarrow rv(\mathsf{m}') \end{cases}$	$\mathcal{E}_k(LR(lv(\mathsf{m}'), rv(\mathsf{m}'), b))$	equal to $lv(\mathsf{m})$
$\mathsf{m} := \mathtt{dec}\ \mathsf{m}', \mathsf{k};\quad \mathsf{k} \in \mathcal{B}(\mathcal{K})$	$\begin{cases} \mathcal{D}_k(lv(\mathsf{m}')), \\ \quad \text{if } lv(\mathsf{m}') \notin qdb_k \\ dec_k^0(lv(\mathsf{m}')), \\ \quad \text{if } lv(\mathsf{m}') \in qdb_k \end{cases}$	$\begin{cases} \mathcal{D}_k(rv(\mathsf{m}')), \\ \quad \text{if } rv(\mathsf{m}') \notin qdb_k \\ dec_k^0(rv(\mathsf{m}')), \\ \quad \text{if } rv(\mathsf{m}') \in qdb_k \end{cases}$
$\mathsf{m} := \mathtt{enc}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \in \mathcal{K} - \mathcal{B}(\mathcal{K})$	$enc(lv(\mathsf{m}'), lv(\mathsf{n}))$	equal to $lv(\mathsf{m})$ $(*)$
$\mathsf{m} := \mathtt{dec}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \in \mathcal{K} - \mathcal{B}(\mathcal{K})$	$dec(lv(\mathsf{m}'), lv(\mathsf{n}))$	$dec(rv(\mathsf{m}'), lv(\mathsf{n}))$ $(*)$
$\mathsf{m} := \mathtt{enc}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \notin \mathcal{K}$	$enc(lv(\mathsf{m}'), lv(\mathsf{n}))$	$enc(rv(\mathsf{m}'), lv(\mathsf{n}))$ $(*)$
$\mathsf{m} := \mathtt{dec}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \notin \mathcal{K}$	$dec(lv(\mathsf{m}'), lv(\mathsf{n}))$	$dec(rv(\mathsf{m}'), lv(\mathsf{n}))$ $(*)$

(*) Note that we are only using $lv(\mathsf{n})$ as the key here.

Table 2.5: Modification of the Operational Semantics with Parameters $\mathsf{s}, \mathcal{K}$ for Key DAGs

ACTION	$v(\mathsf{m})$
`new m;` $\mathsf{m} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	generate nonce
`new m;` $\mathsf{m} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	m_1
$\mathsf{m} := \mathsf{enc}\ \mathsf{m}', \mathsf{k};$ $\mathsf{k} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$enc(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} := \mathsf{dec}\ \mathsf{m}', \mathsf{k};$ $\mathsf{k} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$dec(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} := \mathsf{enc}\ \mathsf{m}', \mathsf{n};$ $\mathsf{n} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$enc(v(\mathsf{m}'), n_0)$
$\mathsf{m} := \mathsf{dec}\ \mathsf{m}', \mathsf{n};$ $\mathsf{n} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$dec(v(\mathsf{m}'), n_0)$

Table 2.6: Operational Semantics of Alternate Simulator for b = 1 with Parameters $\mathsf{s}, \mathcal{K}$ for Key DAGs

- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with s^i: $\mathcal{E}_{s_0^i}(f(s_0, s_0^1, s_0^2, ...))$.

We observe here that $\mathcal{S}$ simulated the execution of another secretive protocol $\mathcal{G}'$ with keys of level $\leq n$ - $s_0^1, s_0^2, ...$ protecting s_0 (operational semantics described in Table 2.6). This is because the root level keys no longer protect the other keys in the DAG - we obtain a transformed DAG with the roots of the earlier DAG removed, and hence of maximum level one less. Therefore, we have:

$$Pr[d = 1|b = 1] - 1/2 = (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta), \tag{2.6}$$

where $\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta)$ is the advantage of $\mathcal{A}$ in breaking the indistinguishability of s against the protocol $\mathcal{G}'$.

By the Equations 2.5 and 2.6 we have:

$$Pr[d = 0|b = 0] - 1/2 = \mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) - (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Theorem 5 (CCA security - Keying) *Assume that a probabilistic poly-time* (s, $\mathcal{K}$)*-adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s}, \mathcal{K})$*-secretive protocol such that there is a key* *graph Γ for $\mathcal{K}$ which is a DAG. Honest principals are allowed to use a key of value $\lambda(\mathsf{s})$ as a key in every trace $\langle e, \lambda \rangle$. The adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key $\lambda(\mathsf{s})$, over the set of all traces $\langle e, \lambda \rangle$, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies IND-CCA key usability for s against $\mathcal{A}$.*

Proof. We will again prove this by induction over the maximum level Γ. If $\mathcal{K}$ consists only of level 0 keys then the result follows from Theorem 2. Suppose the maximum level in Γ is $(n+1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the basis $\mathcal{B}(\mathcal{K})$ of the set of keys $\mathcal{K}$.

We will show that if $\mathcal{A}$ has non-negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key $\lambda(\mathsf{s})$, after the interaction then we can construct either a $|\mathcal{K}'|$-IND-CCA adversary $\mathcal{A}_1$ or contradict the induction hypothesis.

We proceed as in the proof of Theorem 4 to construct the adversary $\mathcal{A}_1$. The only additional operation is that to encrypt or decrypt the term m with s, $\mathcal{S}$ uses s_0 as the key.

In the second phase, $\mathcal{A}_1$ randomly chooses a bit $b' \leftarrow \{0,1\}$. $\mathcal{A}$ sends pairs of messages m_0, m_1 to $\mathcal{A}_1$. $\mathcal{A}_1$ replies with $\mathcal{E}_{s_0}(m_{b'})$. Decryption requests are also served by decrypting with key s_0 ciphertexts not obtained by a query in this phase. $\mathcal{A}$ finishes by outputting a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs $d = 0$ else outputs $d = 1$.

The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}'|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d=0|b=0] - Pr[d=0|b=1] \tag{2.7}$$

The first probability is precisely the probability of $\mathcal{A}$ breaking the 'good-key'-ness of s_0 or equivalently of s. In the case when $b = 1$, the terms were constructed in the following manner:

- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with $\mathsf{k}_i \in \mathcal{K}'$: $\mathcal{E}_{k_i}(f(s_1, s_1^1, s_1^2, ...))$.
- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with s^i: $\mathcal{E}_{s_0^i}(f(s_0, s_0^1, s_0^2, ...))$.

- encrypt term m with s: $\mathcal{E}_{s_0}(m)$.

Again, as in the proof of Theorem 1, terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ values by an extension of Lemma 1 and by Lemmas 2 and 3, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation.

We observe here that $\mathcal{S}$ simulated the execution of another secretive protocol $\mathcal{G}'$ with keys of level $\leq n$ - $s_0^1, s_0^2, \ldots$ protecting s_0. This is because the root level keys no longer protect the other keys in the DAG - we obtain a transformed DAG with the roots of the earlier DAG Γ removed, and hence of maximum level one less. Therefore, we have:

$$Pr[d = 0|b = 1] - 1/2 = (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta) \tag{2.8}$$

By the Equations 2.7 and 2.8 we have:

$$Pr[d = 0|b = 0] - 1/2 = \mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Theorem 6 (CTXT integrity) *Assume that a probabilistic poly-time* $(\mathsf{s}, \mathcal{K})$*- adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s}, \mathcal{K})$*-secretive protocol such that there is a key graph* Γ *for* $\mathcal{K}$ *which is a DAG. In any trace* $\langle e, \lambda\rangle$*, if an honest principal decrypts a ciphertext with a key of value* $\lambda(\mathsf{s})$ *successfully, then with overwhelming probability, over the set of all traces* $\langle e, \lambda\rangle$*, the ciphertext was produced by an honest principal by encryption with* $\lambda(\mathsf{s})$ *if the encryption scheme is IND-CCA and INT-CTXT secure.*

Proof. We will prove this by induction over the maximum level of Γ. If $\mathcal{K}$ consists only of level 0 keys then the result follows from Theorem 3. Suppose the maximum level in Γ is $(n+1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the basis $\mathcal{B}(\mathcal{K})$ of the set of keys $\mathcal{K}$. Suppose during the protocol run, an honest party decrypts a ciphertext with key $\lambda(\mathsf{s})$ successfully which was not produced by an honest party by encryption with $\lambda(\mathsf{s})$. The simulator does not get stuck due to any send, deconstruction or matching action as we have seen in the previous two proofs.

We build a $|\mathcal{K}'|$-IND-CCA adversary $\mathcal{A}_1$ against set of keys $\mathcal{K}'$ along the lines of the proof of Theorem 5. In the course of interaction with $\mathcal{A}$, if $\mathcal{S}$ succeeds in decrypting a ciphertext with key s_0 which was not produced at a previous stage by $\mathcal{S}$ by encryption with s_0, $\mathcal{A}_1$ outputs $d = 0$. Otherwise, it outputs $d = 1$. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}'|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \tag{2.9}$$

Now, $Pr[d = 0|b = 0]$ is the probability of $\mathcal{A}_1$ succeeding in producing a ciphertext with key $\lambda(\mathsf{s})$ which was not obtained through encryption by $\mathcal{A}_1$. $Pr[d = 0|b = 1]$ is the probability of $\mathcal{A}_1$ succeeding in decrypting a ciphertext with level-$(n-1)$ key s_0 (Same argument as in proof of Theorem 5 - the DAG reduces by one level) which was not produced by encryption with s_0. Therefore,

$$Pr[d = 0|b = 0] = \mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) + Pr[d = 0|b = 1]$$

As the encryption scheme is IND-CCA secure, the first probability on the RHS must be negligible. The second probability is negligible due to the induction hypothesis as the encryption scheme is both IND-CCA and INT-CTXT secure. Hence the theorem. $\square$

2.2 Diffie-Hellman

In this section, we formulate a trace property for protocols that use the Diffie-Hellman primitive and prove that, under the Decisional Diffie-Hellman assumption, any protocol that satisfies this condition produces keys that are suitable for keying chosen plaintext (IND-CPA) secure encryption schemes. The motivating application for this result is the fact that many Diffie-Hellman-based key exchange protocols (e.g., IKEv2 [23]) set up keys for use in secure sessions protocols. Such protocols typically provide the desired security with IND-CPA encryption schemes and do not require IND-CCA secure encryption. However, we also state a stronger theorem assuming IND-CCA encryption schemes.

$(\text{keys})_{DH}$	K_{DH}	$::=$	K	keys from table 2.1
			$(g^n)^n$	DH key
$(\text{terms})_{DH}$	t_{DH}	$::=$	t	terms from table 2.1
			g^n	exponentiated term
			K_{DH}	keys
			$t_{DH}.\ t_{DH}$	tuple of terms
			$ENC[K_{DH}](t_{DH})$	term encrypted with key K
$(\text{actions})_{DH}$	a_{DH}	$::=$	a	actions from table 2.1
			$y := \texttt{expg}\ n$	exponentiating a nonce
			$y := \texttt{exp}\ g^n, n$	exponentiating an exponentiated term

Table 2.7: Extension of the Protocol Programming Language with DH Operations

The additional operations for Diffie-Hellman are tabulated in Table 2.7. The group $\mathcal{G}$ and the group element g are fixed, given the security parameter η, and are available to all principals. The bit-strings corresponding to nonces are sampled uniformly randomly from $\mathbb{Z}_{|\mathcal{G}|}$.

Definition 14 (DHSafe Trace) *Let* x *and* y *be two terms of type nonce in a trace* $\langle e, \lambda \rangle$*. We say that* $\langle e, \lambda \rangle$ *is an* (x, y)-DHSafe trace *if the following properties hold for every thread belonging to honest principals:*

- *the thread which generates nonce* x*, ensures that it appears only exponentiated as* g^{x} *in any message sent out. Similarly for* y*.*
- *the thread generating* x *is allowed to generate a key by exponentiating a term* m *such that* $\lambda(\mathsf{m}) = g^{\lambda(\mathsf{y})}$ *to the power* x *and using an appropriate key generation algorithm. However, this key (*g^{xy}*) is only used in the protocol for encrypting messages, not sent as the payload of a message. A similar restriction applies to the thread generating* y*.*

The key generation algorithm referred to in the definition is assumed to map from the uniform distribution of group elements g^n to a distribution computationally indistinguishable from the key distribution required by the symmetric encryption scheme. The second bullet in the definition is not a syntactic condition and can be non-trivial to prove. One usual way protocols achieve this sort of authentication is by using digital signatures. The discussion of proof methods to ensure the conditions required for $DHSafe$-ness and how the results of this chapter fit into a larger context for proving security properties of DH protocols is in Chapter 4.

Definition 15 (DHSafe Protocol) *Let* x *and* y *be two terms of type nonce. Let* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *be a probabilistic poly-time adversary which decides which nonces in a trace to designate* x *and* y*. A protocol* $\mathcal{Q}$ *is an* (x,y)-DHSafe protocol *for* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *if for all sufficiently large security parameters* η*, the probability that a trace* $t(\mathcal{A}_{\mathsf{x},\mathsf{y}}, \mathcal{Q}, \eta)$*, generated by the interaction of* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *with principals following roles of* $\mathcal{Q}$*, is a DHSafe trace with respect to* x *and* y *is overwhelmingly close to 1, the probability being taken over all adversary and protocol randomness. Formally,*

$$(1 - Pr[t(\mathcal{A}_{\mathsf{x},\mathsf{y}}, \mathcal{Q}, \eta) \text{ is DHSafe wrt } \mathsf{x} \text{ and } \mathsf{y}\ ]) \text{ is a negligible function of } \eta$$

As in the case of secretive protocols, here also we implicitly look at the subset of all traces that are DHSafe among all possible traces. Since the set of non-DHSafe traces is a negligible subset of all traces, by definition, any advantage the adversary obtains against the non-DHSafe traces will be cumulatively negligible.

We consider two scenarios involving the usage of the Diffie Hellman key to demonstrate the flexibility of our approach. In the first scenario, the results of decryption using the key do not produce any observable difference on subsequent sends. One useful instance may be when the results of decryption are some non-protocol data that are used internally by the principal. In these cases we make the point that we do not require IND-CCA; IND-CPA is sufficient. In the second scenario, the results of decryption may affect subsequent sends - here we use the IND-CCA assumption.

Theorem 7 (DH-CPA security) *Assume that a probabilistic poly-time adversary* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *interacts with an* (x,y)*-DHSafe protocol. Suppose the encryption scheme used* *by the protocol is IND-CPA secure and the DDH assumption holds for the group containing g. Then the adversary has negligible advantage at winning an IND-CPA game against a symmetric encryption challenger, using the key* k *such that* $\lambda(\mathsf{k}) = keygen(g^{\lambda(\mathsf{x})\lambda(\mathsf{y})})$*, after the interaction provided the results of decryptions with key* k *are not used to construct any message sent out. In other words, the protocol satisfies IND-CPA key usability for* k *against* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *if the results of decryptions with key* k *are not used to construct any message sent out.*

Proof. We will show that if $\mathcal{A}(= \mathcal{A}_{\mathsf{x},\mathsf{y}})$ has non-negligible advantage at winning an IND-CPA game against a symmetric encryption challenger, using the key k, after the interaction then we can construct either a DDH adversary $\mathcal{A}_1$ with non-negligible advantage against DDH in the group containing g or an IND-CPA adversary $\mathcal{A}_2$ with non-negligible advantage against the encryption scheme.

Adversary $\mathcal{A}_1$ is provided, at the outset, with a triple (g^a, g^b, g^c) and has to determine if $c = ab$. It proceeds by simulating the execution of the protocol to adversary $\mathcal{A}$. Following the definition of DHSafe protocols, if an honest principal sends out a message containing x or y, then it has to be constructed from g^{x} or g^{y}. $\mathcal{A}_1$ uses g^a and g^b as the bitstring values of g^{x} and g^{y} respectively. When an honest principal exponentiates a term to the power x or y and generates a key, $\mathcal{A}_1$ uses $k = keygen(g^c)$ as the bitstring value of the key. The operational semantics is described in Table 2.8.

In the second phase, $\mathcal{A}_1$ uniformly randomly chooses a bit b' and provides oracle $\mathcal{E}_k(LR(\cdot,\cdot,b'))$ to $\mathcal{A}$ for an IND-CPA game. $\mathcal{A}$ finishes by outputting a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs 1 else outputs 0. The advantage of $\mathcal{A}_1$ against the DDH challenger is:

$$\mathbf{Adv}_{DDH,\mathcal{A}_1}(\eta) = Pr[\mathcal{A}_1 \ \textit{outputs}\ 1 \mid c = ab] - Pr[\mathcal{A}_1 \ \textit{outputs}\ 1 \mid c \neq ab] \qquad (2.10)$$

Observe that if $c = ab$ then k is the bitstring value $keygen(g^{\lambda(\mathsf{x})\lambda(\mathsf{y})})$. Hence, the first probability is precisely the probability of $\mathcal{A}$ winning an IND-CPA challenge with k as the key after interacting with an (x,y)-*DHSafe protocol.* We will now bound the second probability.

We start by constructing a second adversary $\mathcal{A}_2$ which has access to an encryption oracle $\mathcal{E}_k(LR(\cdot,\cdot,b_1))$ with k randomly generated from the symmetric encryption

Assignment actions:

ACTION	$v(\mathsf{m})$
m in the static list, $\mathsf{m} \notin \mathcal{K}$	value from initial input
`receive m`;	receive from adversary
`new m`; $\quad \mathsf{m} \neq \mathsf{x}, \mathsf{m} \neq \mathsf{y}$	generate nonce
`new m`; $\quad \mathsf{m} = \mathsf{x}$ or $\mathsf{m} = \mathsf{y}$	do nothing
$\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$;	$pair(v(\mathsf{m}'), v(\mathsf{m}''))$
$\mathsf{m} :=$ `fst` m';	$fst(v(\mathsf{m}'))$
$\mathsf{m} :=$ `snd` m';	$snd(v(\mathsf{m}'))$
$\mathsf{m} :=$ `expg` n; $\quad \mathsf{n} \neq \mathsf{x}, \mathsf{n} \neq \mathsf{y}$	$g^{v(\mathsf{n})}$
$\mathsf{m} :=$ `exp` m', n; $\quad \mathsf{n} \neq \mathsf{x}, \mathsf{n} \neq \mathsf{y}$	$v(\mathsf{m}')^{v(\mathsf{n})}$
$\mathsf{m} :=$ `expg` x;	g^a
$\mathsf{m} :=$ `expg` y;	g^b
$\mathsf{m} :=$ `exp` m', y; $\quad v(\mathsf{m}') = g^a$	g^c
$\mathsf{m} :=$ `exp` m', x; $\quad v(\mathsf{m}') = g^b$	g^c
$\mathsf{m} :=$ `exp` m', y; $\quad v(\mathsf{m}') \neq g^a$	stop simulation
$\mathsf{m} :=$ `exp` m', x; $\quad v(\mathsf{m}') \neq g^b$	stop simulation
$\mathsf{m} :=$ `enc` m', k;	$enc(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} :=$ `dec` m', k;	$dec(v(\mathsf{m}'), v(\mathsf{k}))$

Table 2.8: Operational Semantics of the Simulator with Parameters x, y and DDH inputs g^a, g^b, g^c

scheme's key generation algorithm. It has a similar behaviour towards $\mathcal{A}$ as $\mathcal{A}_1$ had except when required to encrypt using the generated key, it queries $\mathcal{E}_k(LR(\cdot,\cdot,b_1))$. Decryption queries are not required as results of decryptions are not used to construct any message sent. In the second phase, $\mathcal{A}_1$ uses the $\mathcal{E}_k(LR(\cdot,\cdot,b_1))$ to provide the IND-CPA challenger to $\mathcal{A}$. $\mathcal{A}$ finishes by outputting a bit d_1 which is what $\mathcal{A}_2$ also outputs. Thus we have:

$$(1/2)\mathbf{Adv}_{IND-CPA,\mathcal{A}_2}(\eta) = Pr[d_1 = b_1] - 1/2 \tag{2.11}$$

Now suppose $\mathcal{A}_2$ instead has access to an encryption oracle $\mathcal{E}_k(LR(\cdot,\cdot,b_1'))$ with k randomly generated from the Diffie-Hellman key generation algorithm. Let d_1' be the output of $\mathcal{A}_2$ and let $\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}$ denote the advantage:

$$(1/2)\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}(\eta) = Pr[d_1' = b_1'] - 1/2 \tag{2.12}$$

The difference $\epsilon(\eta) = Pr[d_1' = b_1'] - Pr[d_1 = b_1]$ is negligible in the security parameter since otherwise $\mathcal{A}_2$ could be used to distinguish between the distribution of keys generated by the two key generation algorithm, which is a contradiction as per our assumption.

We observe here that if $c \neq ab$ for the first LR oracle, it makes no difference to the algorithm $\mathcal{A}$ whether it is interacting with $\mathcal{A}_1$ or $\mathcal{A}_2$.

$$(1/2)\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}(\eta) = Pr[\mathcal{A}_1 \ \mathit{outputs}\ 1 \mid c \neq ab] - 1/2 \tag{2.13}$$

By the equations 2.10, 2.11, 2.12 and 2.13 we have:

$$\begin{aligned}
&Pr[\mathcal{A}\ \mathit{wins\ IND\text{-}CPA\ challenge\ with}\ k\ \mathit{as\ the\ key}\ ] - 1/2 \\
&= Pr[\mathcal{A}_1\ \mathit{outputs}\ 1 \mid c = ab] - 1/2 \\
&= \mathbf{Adv}_{DDH,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}(\eta) \\
&= \mathbf{Adv}_{DDH,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{IND-CPA,\mathcal{A}_2}(\eta) + \epsilon(\eta)
\end{aligned}$$

If the probablity in the LHS is non-negligible, at least one of the advantages in the

RHS must be non-negligible and hence we are done. □

Theorem 8 (DH-CCA security) *Assume that a probabilistic poly-time adversary $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ interacts with an (x,y)-DHSafe protocol. Suppose the encryption scheme used by the protocol is IND-CCA secure and the DDH assumption holds for the group containing g. Then the adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key k such that $\lambda(\mathsf{k}) = keygen(g^{\lambda(\mathsf{x})\lambda(\mathsf{y})})$, after the interaction. In other words, the protocol satisfies IND-CCA key usability for k against $\mathcal{A}_{\mathsf{x},\mathsf{y}}$.*

The proof is almost identical to the proof for theorem 7 with IND-CCA replacing IND-CPA — IND-CCA allows decryption with the key in consideration.

Chapter 3

Inductive Proofs of Computational Secrecy

Present-day Internet users and networked enterprises rely on key management and related protocols that use cryptographic primitives. In spite of the staggering financial value of, say, the total number of credit card numbers transmitted by SSL/TLS in a day, we do not have correctness proofs that respect cryptographic notions of security for many of these relatively simple distributed programs. In light of this challenge, there have been many efforts to develop and use methods for proving security properties of network protocols. Historically, most efforts used an abstract *symbolic model,* also referred to as the *Dolev-Yao* model [56, 41]. More recently, in part to draw stronger conclusions from existing methods and proofs, several groups of researchers have taken steps to connect the symbolic model to probabilistic polynomial-time *computational models* accepted in cryptographic studies, *e.g.* [2, 9, 11, 26, 58, 35, 36, 70].

A fundamental problem in reasoning about secrecy, such as computational indistinguishability of a key from a randomly chosen value, is that such secrecy properties are not trace properties – indistinguishability over a set of possible runs is *not* defined by summing the probability of indistinguishability on each run. As a result, it does not appear feasible to prove computational secrecy properties by induction on the steps of a protocol. A central contribution of this chapter is axiomatizing the trace-based property, *secretive,* described in Chapter 2 in Computational Protocol

Composition Logic (CPCL) [35, 36]. In the process, we generalize a previous induction rule, so that only one core induction principle is needed in the logic. In contrast to proof systems for symbolic secrecy [63, 68], the induction is over actions of honest parties and not the structure of terms. We also extend previous composition theorems [33, 46] to the present setting, and illustrate the power of the resulting system by giving modular formal proofs of authentication and secrecy properties of Kerberos V5 and Kerberos V5 with PKINIT. We are also able to prove properties of a variant of the Needham-Schroeder-Lowe protocol that are beyond the standard rank function method [47, 38].

Our approach may be compared with equivalence-based methods [9, 58, 30, 5], used in [6] to derive some computational properties of Kerberos V5 from a symbolic proof. In equivalence-based methods, the behavior of a symbolic abstraction under symbolic attack must have the same observable behavior as a computational execution under computational (probabilistic polynomial-time) attack. In contrast, our approach only requires an implication between symbolic reasoning and computational execution. While we believe that both approaches have merit, the two are distinguished by (i) the need to additionally prove the absence of a "commitment problem" in [6], which appears to be a fundamental issue in equivalence-based security [31], and (ii) the open problem expressed in [6] of developing compositional methods in that framework. Symbolic abstractions for primitives like Diffie-Hellman key exchange are also problematic for equivalence-based approaches [8, 10], but amenable to treatment in PCL. In contrast to other symbolic or computationally sound methods, PCL reasoning proceeds only over action sequences of the protocol program, yet the conclusions are sound for protocol execution in the presence of attack. This formalizes and justifies a direct reasoning method that is commonly used informally among researchers, yet is otherwise not rigorously connected to reduction arguments.

Section 3.1 describes Computational PCL. The new proof system, and soundness and composition theorems are presented in Section 3.2, and applied in the proofs for Kerberos in Section 3.3.2.

Terms:

N	$::=$	$\hat{X}$	(*name*)
K	$::=$	X	(*key*)
S	$::=$	s	(*session*)
n	$::=$	r	(*nonce*)
T	$::=$	(N, S)	(*thread*)
V	$::=$	x	(*term variable*)
t_B	$::=$	$V \mid K \mid T \mid N \mid n \mid t_B.t_B$	(*basic term*)
t	$::=$	$t_B \mid E_{sym}[K](t) \mid E_{pk}[K](t) \mid t.t$	(*term*)

Actions:

`a`	$::=$	
	$\mid$	`new` n
	$\mid$	$V :=$ `symenc` t, K
	$\mid$	$V :=$ `symdec` t, K
	$\mid$	$V :=$ `pkenc` t, K
	$\mid$	$V :=$ `pkdec` t, K
	$\mid$	`match` t/t
	$\mid$	`send` t
	$\mid$	`receive` V

Table 3.1: Syntax of protocol terms and actions

3.1 Computational PCL

We begin by reviewing a protocol notation, protocol logic, and a security model for key exchange developed in earlier work [33, 35, 36].

3.1.1 Protocol Syntax

We use a simple "protocol programming language" based on [40, 32, 33] to represent a protocol by a set of roles, such as "Initiator", "Responder" or "Server", each specifying a sequence of actions to be executed by a honest participant. The syntax of terms and actions is given in Table 3.1.

Names, sessions and threads: We use $\hat{X}, \hat{Y}, \ldots$ as *names* for protocol participants. Since a particular participant might be involved in more than one session at a time, we will give unique names to sessions and use $(\hat{X}, s)$ to designate a particular *thread* being executed by $\hat{X}$. All threads of a participant $\hat{X}$ share the same asymmetric key denoted by X. As a notational convenience, we will sometimes write $\tilde{X}$ for an arbitrary thread of $\hat{X}$.

Terms, actions, and action lists: Terms name messages and their parts, such as nonces, keys, variables and pairs. For technical reasons, we distinguish *basic terms* from *terms* that may contain encryption. To account for probabilistic encryption, encrypted terms explicitly identify the randomness used for encryption. Specifically, $\{t\}_K^n$ indicates the encryption of t with key K using randomness n generated for the purpose of encryption. We write $m \subseteq m'$ when m is a subterm of $m' \in t$.

Action Predicates:
$\mathsf{a} \quad ::= \quad \mathsf{Send}(T,t) \,|\, \mathsf{Receive}(T,t) \,|\, \mathsf{New}(T,n)$

Formulas:
$\varphi \quad ::= \quad \mathsf{a} \,|\, t = t \,|\, \mathsf{Start}(T) \,|\, \mathsf{Possess}(T,\, t) \,|\, \mathsf{Indist}(T,\, t) \,|\, \mathsf{Honest}(N) \,|$
$\qquad \mathsf{Start}(T) \,|\, \varphi \wedge \varphi \,|\, \varphi \vee \varphi \,|\, \exists Var.\, \varphi \,|\, \forall Var.\, \varphi \,|\, \neg\varphi \,|\, \varphi \supset \varphi \,|\, \varphi \Rightarrow \varphi$

Modal formulas:
$\Psi \quad ::= \quad \varphi\ [Strand]_T\ \varphi$

Table 3.2: Syntax of the logic

Actions include nonce generation, encryption, decryption, pattern matching, and communication steps (sending and receiving). An *ActionList* consists of a sequence of actions that contain only basic terms. This means that encryption cannot be performed implicitly; explicit enc actions, written as assignment, must be used instead. We assume that each variable will be assigned at most once, at its first occurrence. For any $s \in ActionList$, we write $s_{|X}$ to denote the subsequence of s containing only actions of a participant (or a thread) X.

Strands, roles, protocols and execution: A *strand* is an *ActionList*, containing actions of only one thread. Typically we will use notation $[ActionList]_{\tilde{X}}$ to denote a strand executed by thread $\tilde{X}$ and drop the thread identifier from the actions themselves. A *role* is a strand together with a basic term representing the initial knowledge of the thread. A *protocol* is a finite set of *Roles*, together with a basic term representing the initial intruder knowledge.

An *execution strand* is a pair $ExecStrand ::= InitialState(\mathcal{I}); ActionList$ where $\mathcal{I}$ is a data structure representing the initial state of the protocol, as produced by the initialization phase from Section 3.1.5. In particular, this includes the list of agents and threads, the public/private keys and honesty/dishonesty tokens of each agent, and the roles played by each thread.

3.1.2 Logic Syntax

The syntax of formulas is given in Table 3.2. Protocol proofs will usually use modal formulas of the form $\psi[P]_{\tilde{X}}\varphi$. Most formulas have the same intuitive meaning in the computational semantics as in the symbolic model [32, 33], except for predicates Possess and Indist. We summarize the meaning of formulas informally below, with precise semantics in the next section.

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathsf{Send}(\tilde{X}, t)$ holds in a run where the thread $\tilde{X}$ has sent the term t. , while $\mathsf{Honest}(\hat{X})$ means that $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol.

In the symbolic model [32, 33],the predicate Has states that a principal can "derive" a message or its contents from the information gathered during protocol execution. We use $\mathsf{Possess}(\tilde{X}, t)$ to state that it is possible to derive t by Dolev-Yao rules from $\tilde{X}$'s view of the run and $\mathsf{Indist}(\tilde{X}, t)$ to state that no probabilistic polynomial-time algorithm, given $\tilde{X}$'s view of the run, can distinguish t from a random value from the same distribution. Typically, we use Possess to say that some honest party obtained some secret, and Indist to say that the attacker does not have any partial information about a secret.

3.1.3 Proof System

A representative fragment of the axioms and inference rules in the proof system are collected in Table 3.3. For expositional convenience, we divide the axioms into three groups.

The axioms about protocol actions state properties that hold in the state reached by executing one of the actions in a state in which formula ϕ holds. Note that the a in axiom **AA1** is any one of the actions and a is the corresponding predicate in the logic. Axiom **A1** states that two different threads cannot generate the same nonce while axiom **A2** states that if a thread encrypts a message with a key, it possesses both the message and the key.

The possession axioms reflect a fragment of Dolev-Yao rules for constructing or

Axioms for protocol actions

AA1	$\phi[a]_X\ \mathsf{a}$
AA2	$\mathsf{Start}(X)[\,]_X\ \neg\mathsf{a}(\mathsf{X})$
AA3	$\neg\mathsf{Send}(X,t)[b]_X\neg\mathsf{Send}(X,t)$ if $\sigma\mathsf{Send}(X,t) \neq \sigma\mathsf{b}$ for all substitutions σ
ARP	$\mathsf{Receive}(X,p(x))[\mathtt{match}\ q(x)\ \mathtt{as}\ q(t)]_X\ \mathsf{Receive}(X,p(t))$
P1	$\mathsf{Persist}(X,t)[a]_X\ \mathsf{Persist}(X,t)$, for $\mathsf{Persist} \in \{\mathsf{Has},\mathsf{Send},\mathsf{Receive}\}$
A1	$\mathsf{New}(X,n) \wedge \mathsf{New}(Y,n) \supset X = Y$
A2	$\mathsf{SymEnc}(X,m,k) \supset \mathsf{Possess}(X,k) \wedge \mathsf{Possess}(X,m)$

Possession Axioms

ORIG	$\mathsf{New}(X,x) \supset \mathsf{Possess}(X,x)$
TUP	$\mathsf{Possess}(X,x) \wedge \mathsf{Possess}(X,y) \supset \mathsf{Possess}(X,x.y)$
REC	$\mathsf{Receive}(X,x) \supset \mathsf{Possess}(X,x)$
PROJ	$\mathsf{Has}(X,x.y) \supset \mathsf{Possess}(X,x) \wedge \mathsf{Possess}(X,y)$

Generic Rules

$$\frac{\theta[P]_X\phi \quad \theta[P]_X\psi}{\theta[P]_X\phi \wedge \psi}\ \mathbf{G1} \qquad \frac{\theta' \supset \theta \quad \theta[P]_X\phi \quad \phi \supset \phi'}{\theta'[P]_X\phi'}\ \mathbf{G2} \qquad \frac{\phi}{\theta[P]_X\phi}\ \mathbf{G3}$$

Table 3.3: Fragment of the Proof System

decomposing messages while the encryption axioms symbolically model encryption. The generic rules are used for manipulating modal formulas.

3.1.4 The Honesty Rule

The honesty rule is essentially an invariance rule for proving properties of all roles of a protocol. It is similar to the basic invariance rule of LTL [55]. The honesty rule is used to combine facts about one role with inferred actions of other roles.

For example, suppose Alice receives a response from a message sent to Bob. Alice may wish to use properties of Bob's role to reason about how Bob generated his reply. In order to do so, Alice may assume that Bob is honest and derive consequences from this assumption. Since honesty, by definition in this framework, means "following one or more roles of the protocol," honest principals must satisfy every property that is a provable invariant of the protocol roles.

Recall that a protocol $\mathcal{Q}$ is a set of roles, $\mathcal{Q} = \{\rho_1, \rho_2, \ldots, \rho_k\}$. If $\rho \in \mathcal{Q}$ is a role of protocol $\mathcal{Q}$, we write $P \epsilon BS(\rho)$ if P is a continuous segment of the actions of role ρ such that (a) P is the empty sequence; or (b) P starts at the beginning of ρ and goes up to the first receive ; or (c) P starts from a receive action and goes up to the next receive action; or (d) P starts from the last receive action and continues till the end of the role. We call such a P a *basic sequence* of role ρ. The reason for only considering segments starting from a read and continuing till the next read is that if a role contains a send, the send may be done asynchronously without waiting for another role to receive. Therefore, we can assume without loss of generality that the only "pausing" states of a principal are those where the role is waiting for input. If a role calls for a message to be sent, then we dictate that the principal following this role must complete the send before pausing.

Since the honesty rule depends on the protocol, we write $\mathcal{Q} \vdash \theta[P]_X\phi$ if $\theta[P]_X\phi$ is provable using the honesty rule for $\mathcal{Q}$ and the other axioms and proof rules.

$$\frac{[\,]_X\ \phi \quad \forall \rho \in \mathcal{Q}. \forall P \epsilon BS(\rho).\ \phi\ [P]_X\ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(\hat{X}) \supset \phi}\ \textbf{HON} \qquad \begin{array}{l}\text{no free variable}\\ \text{in } \phi \text{ except } X\\ \text{bound in } [P]_X\end{array}$$

In words, if ϕ holds at the beginning of every role of $\mathcal{Q}$ and is preserved by all its basic sequences, then every honest principal executing protocol $\mathcal{Q}$ must satisfy ϕ. The side condition prevents free variables in the conclusion $\mathsf{Honest}(\hat{X}) \supset \phi$ from becoming bound in any hypothesis. Intuitively, since ϕ holds in the initial state and is preserved by all basic sequences, it holds at all pausing states of any run.

3.1.5 Protocol Execution

Given a protocol, adversary, and value of the security parameter, we define a set of protocol traces, each associated with the random bits that produce this sequence of actions and additional randomness for algorithms used in the semantics of formulas about the run. The definition proceeds in two phases. In the initialization phase, we assign a set of roles to each principal, identify a subset which is honest, and provide all entities with private-public key pairs and random bits. In the execution phase, the adversary executes the protocol by interacting with honest principals, as in the accepted cryptographic model of [16].

Initialization: We fix the protocol Q, adversary A, security parameter η, and some randomness R of size polynomially bounded in η. Each principal and each thread (*i.e.*, an instance of a protocol role executed by the principal) is assigned a unique bitstring identifier. We choose a sufficiently large polynomial number of bitstrings $i \in I \subseteq \{0,1\}^\eta$ to represent the names of principals and threads. Randomness R is split into r_i for each honest $i \in I$ (referred to as "coin tosses of honest party i") and R_A (referred to as "adversarial randomness").

The adversary designates some of the principals as *honest* and the rest of the principals as *dishonest*. Intuitively, honest principles will follow one or more roles of the protocol faithfully. The adversary chooses a set of threads, and to each thread it assigns a strand (a program to be executed by that thread), under the restriction that all threads of honest principals are assigned roles of protocol Q.

The key generation algorithm $\mathcal{K}$ of a public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is run on $\mathbf{1}^\eta$ for each participant a using randomness r_a, and producing a public-private key pair (pk_a, sk_a). The public key pk_a is given to all participants and to the adversary A; the private key is given to all threads belonging to this principal and to the adversary if the principal is dishonest.

Generating Computational Traces: Following [16], we view an agent i trying to communicate with agent j in protocol session s as a (stateful) oracle $\Pi^s_{i,j}$. The state of each oracle is defined by a mapping λ from atomic symbols to bitstrings (with variables and nonces renamed to be unique for each role) and a counter c. Each oracle proceeds to execute a step of the protocol as defined by actions in the corresponding role's action list, when activated by the adversary.

We omit the details of communication between the adversary and the oracles, and focus on computational interpretation of symbolic protocol actions. Let a_c be the current action in the *ActionList* defining some role of participant i in session s, *i.e.*, $Thread = (i', s')$ where $i = \lambda(i'), s = \lambda(s')$.

If $a_c = (\texttt{new}\ (i', s'), v)$, then update λ so that $\lambda(v) = NonceGen(R_i)$, where $NonceGen$ is a nonce generation function(*e.g.*, $NonceGen$ simply extracts a fresh piece of R_i). If $a_c = (v := \texttt{enc}\ \ (i', s'), j, u)$, then update λ so that $\lambda(v) = \mathcal{E}(\lambda(u), pk_j, R_i)$ where $\mathcal{E}(\lambda(u), pk_j, R_i)$ is the result of executing the public-key encryption algorithm on plaintext $\lambda(u)$ with public key pk_j and fresh randomness extracted from R_i. For brevity, we omit computational interpretation of decryption and matching (pairing, unpairing, and equality-test) actions. Sending a variable $\texttt{send}\ \ (i', s'), v$ is executed by sending $\lambda(v)$ to the adversary, and receiving $\texttt{receive}\ \ (i', s'), v$ is executed by updating λ so that $\lambda(v) = m$ where m is the bitstring sent by the adversary.

At any time during the protocol execution, the adversary A may record any internal, private message on a special *knowledge tape*. This tape is not read by any participant of the protocol. However, its content will be made available to the test algorithms used to decide if a given security formula containing $\mathsf{Indist}(...)$ is valid or not. Let K be $[(i_1, m_1), .., (i_n, m_n)]$ the list of messages m_k written by A on the knowledge tape, indexed by the number of actions i_k already executed when m_k was written (position in the protocol execution). This index will be useful to remember a previous state of the knowledge tape.

At the end of the protocol execution, the adversary A outputs a pair of integers (p_1, p_2) on an *output tape*. When the security formula is a modal formula $\theta[P]_X\varphi$, these two integers represent two positions in the protocol execution where the adversary claims that the formula is violated, i.e. that θ is true in p_1 but φ is false in p_2, with P between p_1 and p_2. Let O be this pair (p_1, p_2) of integers written on the output tape.

The symbolic trace of the protocol is the execution strand $e \in ExecStrand$ which lists, in the order of execution, all honest participant actions and the dishonest participant's `send` and `receive` actions. This strand contains two parts: $InitialState(\mathcal{I})$ stores the initialization data, and the rest is an ordered list of all exchanged messages and honest participants' internal actions.

Definition 16 *(Computational Traces) Given a protocol Q, an adversary A, a security parameter η, and a sequence of random bits $R \in \{0,1\}^{p(\eta)}$ used by the honest principals and the adversary, a* run *of the protocol is the tuple $\langle e, \lambda, O, K, R\rangle$ where e is the symbolic execution strand, $\lambda : Term(e) \to \{0,1\}^{p(\eta)}$ maps the symbolic terms in e to bitstrings, O is the pair of integers written on the output tape, and K is the indexed list of messages written on the knowledge tape. Finally, $p(x)$ is a polynomial in x.*

A computational trace *is a run with two additional elements: $R_T \in \{0,1\}^{p(\eta)}$, a sequence of random bits used for testing indistinguishability, and $\sigma : FVar(\varphi) \to \{0,1\}^{p(\eta)}$, a substitution that maps free variables in a formula to bitstrings. The set of computational traces is*

$$T_Q(A,\eta) = \{\langle e, \lambda, O, K, R, R_T, \sigma\rangle \mid R, R_T \textit{ chosen uniformly}\}.$$

Definition 17 *(Participant's View) Given a protocol Q, an adversary A, a security parameter η, a participant $\tilde{X}$ and a trace $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T_Q(A,\eta)$, $View_t(\tilde{X})$ represents $\tilde{X}'s$ view of the trace. It is defined precisely as follows:*

If $\hat{X}$ is honest, then $View_t(\tilde{X})$ is the initial knowledge of $\tilde{X}$, a representation of $e_{|\tilde{X}}$ and $\lambda(x)$ for any variable x in $e_{|\tilde{X}}$. If $\hat{X}$ is dishonest, then $View_t(\tilde{X})$ is the union of the knowledge of all dishonest participants $\tilde{X}'$ after the trace t (where $View_t(\tilde{X}')$ is defined as above for honest participants) plus K, the messages written on the knowledge tape by the adversary.

The following three definitions are used in the semantics of the predicate $\mathsf{Indist}()$. Informally, based on some trace knowledge K, the distinguisher D tries to determine which of two bitstrings is the value of a symbolic term. One of the bitstrings will be the computational value of the term in the current run, while the other will be a random bitstring of the same structure, chosen in a specific way. The order of the two bitstrings presented to the distinguisher is determined by an LR Oracle using a random selector bit.

Definition 18 *(LR Oracle) The LR Oracle [13] is used to determine which of two bitstrings are presented depending on the value of the selector bit, i.e.* $LR(s_0, s_1, b) = s_b$.

Definition 19 *(Distinguishing test input) Let* u *be a symbolic term and* σ *be a substitution that maps variables of* u *to bitstrings. We construct another bitstring* $f(u, \sigma, r)$*, whose symbolic representation is the same as* u*. Here,* r *is a sequence of bits chosen uniformly at random. The function* f *is defined by induction over the structure of the term* u.

- *Nonce* u : $f(u, \sigma, r) = r$
- *Name/Key* u : $f(u, \sigma, r) = \sigma(u)$
- *Pair* $u = \langle u_1, u_2 \rangle$: $f(\langle u_1, u_2 \rangle, \sigma, r_1; r_2) = \langle f(u_1, \sigma, r_1), f(u_2, \sigma, r_2) \rangle$
- *Encryption* $u = \{v\}_K^n$: $f(\{v\}_K^n, \sigma, r_1; r_2) = \mathcal{E}(f(v, \sigma, r_1), \sigma(K), r_2)$

Definition 20 *(Distinguisher) A distinguisher* D *is a polynomial time algorithm which takes as input a tuple* $\langle K, t, \langle s_0, s_1 \rangle, R, \eta \rangle$*, consisting of knowledge* K*, symbolic term* t*, two bitstrings* s_0 *and* s_1*, randomness* R *and the security parameter* η*, and outputs a bit* b'.

The next definition is used while defining semantics of modal formulas. Given a set T of traces and a strand P of actions executed by a thread $\tilde{X}$, the set T_P includes only those traces from T which contain P. $Pre(T_P)$ is obtained from T_P by taking the initial segment of each trace upto the point where P starts. The precondition of a modal formula is evaluated over this set. $Post(T_P)$ is similarly defined; the only difference is now the trace is cut at the point that P ends. The postcondition of a modal formula is evaluated over this set. The begin and end positions are determined by the component O in the trace.

Definition 21 *(Splitting computational traces) Let T be a set of computational traces and $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$. $O = \langle p_1, p_2\rangle$, $e = InitialState(\mathcal{I}); s$, and $s = s_1; s_2; s_3$ with p_1, p_2 the start and end positions of s_2 in s. Given a strand P executed by participant $\tilde{X}$, we denote by T_P the set of traces in T for which there exists a substitution σ' which extends σ to variables in P such that $\sigma'(P) = \lambda(s_{2|\tilde{X}})$. The complement of this set is denoted by $T_{\neg P}$ and contains all traces which do not have any occurrence of the strand P. We define the set of traces $Pre(T_P) = \{t[s \leftarrow s_1, K \leftarrow K_{\leq p_1}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$, where $K_{\leq p}$ is the restriction of the knowledge tape K to messages written before the position p. We define the set of traces $Post(T_P) = \{t[s \leftarrow s_1; s_2, K \leftarrow K_{\leq p_2}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$.*

3.1.6 Computational Semantics

The semantics of a formula φ on a set T of computational traces is a subset $T' \subseteq T$ that respects φ in some specific way. For many predicates and connectives, the semantics is essentially straightforward. For example, an action predicate such as Send selects a set of traces in which a send occurs. However, the semantics of predicates Indist and $\mathsf{Possess}$ is inherently more complex.

Intuitively, an agent possesses the value of an expression (such as another agent's nonce or key) if the agent can compute this value from information it has seen, with high probability. If an agent is honest, and therefore follows the rules of the protocol, then it suffices to use a simple, symbolic algorithm for computing values from information seen in the run of a protocol. For dishonest agents, we would prefer in principle to allow any probabilistic polynomial-time algorithm. However, quantifying over such algorithms, in a way that respects the difference between positive and negative occurrences of the predicate in a formula, appears to introduce some technical complications. Therefore, in the interest of outlining a relatively simple form of computational semantics, we will use a fixed algorithm. This gives a useful semantics for formulas where $\mathsf{Possess}(\tilde{X}, u)$ is used under the hypothesis that $\hat{X}$ is honest. We leave adequate treatment of the general case for future work.

Intuitively, an agent has partial information about the value of some expression if the agent can distinguish that value, when presented, from a random value generated according to the same distribution. More specifically, an agent has partial information about a nonce u if, when presented with two bitstrings of the appropriate length, one the value of u and the other chosen randomly, the agent has a good chance of telling which is which. As with $\mathsf{Possess}$, there are technical issues associated with positive and negative occurrences of the predicate. For positive occurrences of Indist, we should say that *no* probabilistic polynomial-time algorithm has more than a negligible chance, where as for $\neg\mathsf{Indist}(\ldots)$ we want to say that *there exists* a probabilistic polynomial-time distinguisher. In order to present a reasonably understandable semantics, and establish a useful basis for further exploration of computational semantics of symbolic security logics, we give an interpretation that appears accurate for formulas that have only positive occurrences of Indist and could be somewhat anomalous for formulas that contain negative occurrences. This seems adequate for reasoning about many secrecy properties, since these are expressed by saying that at the end of any run of the protocol, a value used in the run is indistinguishable from random.

Conditional implication $\theta \Rightarrow \varphi$ is interpreted using the negation of θ and the conditional probability of φ given θ. This non-classical interpretation of implication seems to be essential for relating provable formulas to cryptographic-style reductions involving conditional probabilities. On the other hand, $\Rightarrow$ coincides with $\supset$ in formulas where Indist does not appear on the right hand size of the implication.

We inductively define the semantics $[\![\,\varphi\,]\!](T, D, \epsilon)$ of a formula φ on the set T of traces, with distinguisher D and tolerance ϵ. The distinguisher and tolerance are not used in any of the clauses except for Indist, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value. In definition 22 below, the tolerance is set to a negligible function of the security parameter and $T = T_Q(A, \eta)$ is the set of traces of a protocol Q with adversary A.

- $[\![\,\mathsf{Send}(\tilde{X}, u)\,]\!](T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that some action in the symbolic execution strand e has the form $\texttt{send}\ \ \tilde{Y}, v$ with $\lambda(\tilde{Y}) = \sigma(\tilde{X})$ and $\lambda(v) = \sigma(u)$. Recall that σ maps formula variables to bitstrings and represents the environment in which the formula is evaluated.

- $[\![\, \mathsf{a}(\,\cdot\,,\,\cdot\,) \,]\!](T, D, \epsilon)$ for other action predicates a is similar to $\mathsf{Send}(\tilde{X}, u)$.

- $[\![\, \mathsf{Honest}(\hat{X}) \,]\!](T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ where $e = \mathit{InitialState}(\mathcal{I}); s$ and $\sigma(X)$ is designated *honest* in the initial configuration $\mathcal{I}$. Since we are only dealing with static corruptions in this dissertation, the resulting set is either the whole set T or the empty set ϕ depending on whether a principal is honest or not.

- $[\![\, \mathsf{Start}(\tilde{X}) \,]\!](T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ where $e = \mathit{InitialState}(\mathcal{I}); s$ and $\lambda(s)_{|\sigma(\tilde{X})} = \epsilon$. Intuitively, this set contains traces in which $\tilde{X}$ has executed no actions.

- $[\![\, \theta \wedge \varphi \,]\!](T, D, \epsilon) \; = \; [\![\, \theta \,]\!](T, D, \epsilon) \, \cap \, [\![\, \varphi \,]\!](T, D, \epsilon)$.

- $[\![\, \theta \vee \varphi \,]\!](T, D, \epsilon) \; = \; [\![\, \theta \,]\!](T, D, \epsilon) \, \cup \, [\![\, \varphi \,]\!](T, D, \epsilon)$.

- $[\![\, \neg\varphi \,]\!](T, D, \epsilon) \; = \; T \setminus [\![\, \varphi \,]\!](T, D, \epsilon)$.

- $[\![\, \exists x.\, \varphi \,]\!](T, D, \epsilon) \; = \; \bigcup_{\beta}([\![\, \varphi \,]\!](T[x \leftarrow \beta], D, \epsilon)[x \leftarrow \sigma(x)])$
  with $T[x \leftarrow \beta] = \{t[\sigma[x \leftarrow \beta]] \mid t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T\}$, and β any bitstring of polynomial size.

- $[\![\, \theta \Rightarrow \varphi \,]\!](T, D, \epsilon) \; = \; [\![\, \neg\theta \,]\!](T, D, \epsilon) \, \cup \, [\![\, \varphi \,]\!](T', D, \epsilon)$, where $T' = [\![\, \theta \,]\!](T, D, \epsilon)$. Note that the semantics of φ is taken over the set T' given by the semantics of θ, as discussed earlier in this section.

- $[\![\, u = v \,]\!](T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that $\sigma(u) = \sigma(v)$.

- $[\![\, \mathsf{Possess}(\tilde{X},\, u) \,]\!](T, D, \epsilon)$ includes all traces $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that $\sigma(u)$ can be built from $\mathit{View}_t(\sigma(\tilde{X}))$ with the Dolev-Yao deduction rules.

- $[\![\, \mathsf{Indist}(\tilde{X},\, u) \,]\!](T, \epsilon, D) = T$ if

$$\frac{|\{D(\mathit{View}_t(\sigma(\tilde{X}))), u, LR(\sigma(u), f(u, \sigma, r), b), R_D, \eta) = b \mid t \in T\}|}{|T|} \leq \frac{1}{2} + \epsilon$$

and the empty set ϕ otherwise. Here, the random sequence $b; r; R_D = R_T$, the testing randomness for the trace t.

- $[\![\theta[P]_{\tilde{X}}\varphi]\!](T, D, \epsilon) = T_{\neg P} \cup [\![\neg\theta]\!](Pre(T_P), D, \epsilon) \cup [\![\varphi]\!](Post(T_P), D, \epsilon)$ with $T_{\neg P}$, $Pre(T_P)$, and $Post(T_P)$ as given by Definition 21.

Definition 22 *A protocol Q satisfies a formula φ, written $Q \models \varphi$, if $\forall A$ providing an active protocol adversary, $\forall D$ providing a probabilistic-polynomial-time distinguisher, $\forall \nu$ giving a negligible function, $\exists N$, $\forall \eta \geq N$,*

$$|\, [\![\varphi]\!](T, D, \nu(\eta)) \,| \ / \ |\, T \,| \ \geq \ 1 - \nu(\eta)$$

where $[\![\varphi]\!](T, D, \nu(\eta))$ is the subset of T given by the semantics of φ and $T = T_Q(A, \eta)$ is the set of computational traces of protocol Q generated using adversary A and security parameter η, according to Definition 16.

3.2 Proof System for Secretive Protocols

In this section, we present a general induction rule, axiomatize the mathematical definition of a *secretive protocol* given in chapter 2 and formulate axioms stating that *secretive protocols* guarantee certain computational properties. The soundness proofs of these axioms are based on the theorems in Chapter 2.

3.2.1 Establishing Secretive Protocols

We introduce the predicate $\mathsf{Good}(X, m, s, \mathcal{K})$ to assert that the thread X constructed the term m in accordance with the rules allowing a *secretive protocol* with respect to nonce s and set of keys $\mathcal{K}$ to send out m. More formally, $[\![\mathsf{Good}(X, m, s, \mathcal{K})]\!](T, D, \epsilon)$ is the collection of all traces $t \in T$ where thread X constructs the term m in a ‘good’ way. Received messages, data of atomic type different from nonce or key, nonces different from s are all ‘good’ terms. Constructions that are ‘good’ consist of pairing or unpairing good terms, encrypting good terms, encrypting any term with a key in

$\mathcal{K}$ and decrypting good terms with keys not in $\mathcal{K}$. The following axioms formalize reasoning about the Good predicate by induction on actions in protocol roles.

G0 $\mathsf{Good}(X, a, s, \mathcal{K})$, if a is of an atomic type different from nonce

G1 $\mathsf{New}(Y, n) \wedge n \neq s \supset \mathsf{Good}(X, n, s, \mathcal{K})$

G2 $[\texttt{receive}\ m;]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$

G3 $\mathsf{Good}(X, m, s, \mathcal{K})\ [\mathsf{a}]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$, for all actions a

G4 $\mathsf{Good}(X, m, s, \mathcal{K})\ [\texttt{match}\ m\ \texttt{as}\ m';]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

G5 $\mathsf{Good}(X, m_0, s, \mathcal{K}) \wedge \mathsf{Good}(X, m_1, s, \mathcal{K})\ [m := \texttt{pair}\ m_0, m_1;]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$

G6 $\mathsf{Good}(X, m, s, \mathcal{K})\ [m' := \texttt{symenc}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

G7 $k \in \mathcal{K}\ [m' := \texttt{symenc}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

G8 $\mathsf{Good}(X, m, s, \mathcal{K}) \wedge k \notin \mathcal{K}\ [m' := \texttt{symdec}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

In the following lemma, the additional field σ in the trace definition refers to an environment that maps free variables in a formula to bitstrings.

Lemma 4 *If* $\mathsf{Good}(X, m, s, \mathcal{K})$ *holds for a trace* $\langle e, \lambda, \cdots, \sigma\rangle$*, then any bilateral simulator with parameters* $s, \mathcal{K}$*, executing symbolic actions* e *produces identical bitstring representations for* m *on both sides of the simulation,* i.e., *we will have* $lv(m) = rv(m)$*.*

Proof. The proof is by induction on the construction of 'good' terms. The base cases for received messages, data of atomic type different from nonce or key simply follows from the operational semantics of the simulator. For encryption of a good term and decryption with a key not in $\mathcal{K}$, we use the fact that only the $lv()$ of the key is used in the operational semantics for encryption and decryption, hence the result also has equal $lv()$ and $rv()$ values. The case for encryption of any term with a key in $\mathcal{K}$ follows as the operational semantics for this case produces the same value for $lv()$ and $rv()$ in the result. $\square$

The formula $\mathsf{SendGood}(X, s, \mathcal{K})$ asserts that all messages that thread X sends out are good and $\mathsf{Secretive}(s, \mathcal{K})$ asserts that all honest threads only send out good messages. Formally,

$$\begin{aligned}\mathsf{SendGood}(X,s,\mathcal{K}) &\equiv \forall m.\ (\mathsf{Send}(X,m) \supset \mathsf{Good}(X,m,s,\mathcal{K}))\\ \mathsf{Secretive}(s,\mathcal{K}) &\equiv \forall X.\ (\mathsf{Honest}(\hat{X}) \supset \mathsf{SendGood}(X,s,\mathcal{K}))\end{aligned}$$

The axioms $\mathbf{SG0-2}$ are based on the definition of SendGood:

SG0 $\mathsf{Start}(X)\ [\,]_X\ \mathsf{SendGood}(X,s,\mathcal{K})$

SG1 $\mathsf{SendGood}(X,s,\mathcal{K})\ [\mathsf{a}]_X\ \mathsf{SendGood}(X,s,\mathcal{K})$, where a is not a send.

SG2 $\mathsf{SendGood}(X,s,\mathcal{K})\ [\texttt{send}\ m;]_X\ \mathsf{Good}(X,m,s,\mathcal{K}) \supset \mathsf{SendGood}(X,s,\mathcal{K})$

SG1 is obviously valid for nonce generation, message receipt, encryption and pairing actions. Soundness for unpairing and decryption requires consistency of deconstructions in the bilateral simulation, e.g. unpairing should succeed on one side iff it succeeds on the other. Soundness of **SG2** follows from the operational semantics of the simulator on a send action and Lemma 4.

The $\mathbf{IND}_{GOOD}$ rule which follows states that if all honest threads executing some basic sequence in the protocol locally construct good messages to be sent out, given that they earlier also did so, then we can conclude $\mathsf{Secretive}(s,\mathcal{K})$.

$$\mathbf{IND}_{GOOD}\quad \frac{\forall\rho\in\mathcal{Q}.\forall P\in BS(\rho).\ \ \mathsf{SendGood}(X,s,\mathcal{K})\ [P]_X\ \Phi \supset \mathsf{SendGood}(X,s,\mathcal{K})}{\mathcal{Q}\vdash\Phi\supset\mathsf{Secretive}(s,\mathcal{K})}\ (*)$$

$(*)$: $[P]_X$ does not capture free variables in Φ, $\mathcal{K}$, s, and Φ is a prefix closed trace formula.

A set of basic sequences (BS) of a role is any partition of the sequence of actions in a role such that if any element sequence has a `receive` then it is only at its beginning. The formula Φ has to be *prefix closed* which means that it is a formula such that if it is true at some point in a trace, it is also true at all earlier points. This rule is an instance of a more general induction rule **IND** which is obtained by replacing $\mathsf{SendGood}(X,s,\mathcal{K})$ by a general trace formula $\Psi(X)$ and requiring that $\mathsf{Start}(X)\ [\ ]_X\ \Phi \supset \Psi(X)$. The instance of the latter formula, the base case of the induction, is trivially satisfied when $\Psi(X)$ is $\mathsf{SendGood}(X, s, \mathcal{K})$ because of axiom **SG0**.

The predicate $\mathsf{ContainsOpen}(m, a)$ asserts that a can be obtained from m by a series of unpairings only. Formally,

CO0 $\mathsf{ContainsOpen}(a, a)$

CO1 $\mathsf{ContainsOpen}(m_0.m_1, a) \equiv \mathsf{ContainsOpen}(m_0, a) \vee \mathsf{ContainsOpen}(m_1, a)$

3.2.2 Relating Secretive Protocols to Good Keys

The remaining axioms relate the concept of a secretive protocol, which is trace-based, to complexity theoretic notions of security. As defined in Chapter 2, a level-0 key is only used as a key. Note that this is a syntactic property and is evident from inspection of the protocol roles. Typically, a long-term key shared by two principals is level-0. A nonce is established to be a level-1 key when the protocol is proved to be a *secretive protocol* with respect to the nonce and a set of level-0 keys. This concept is extended further to define level-2 keys.

The formula $\mathsf{InInitSet}(X, s, \mathcal{K})$ asserts X is either the generator of nonce s or a possessor of some key in the basis $\mathcal{B}(\mathcal{K})$. $\mathsf{GoodInit}(s, \mathcal{K})$ asserts that all such threads belong to honest principals. Formally,

$$\mathsf{InInitSet}(X, s, \mathcal{K}) \equiv \exists k \in \mathcal{B}(\mathcal{K}).\, \mathsf{Possess}(X, k) \vee \mathsf{New}(X, s)$$
$$\mathsf{GoodInit}(s, \mathcal{K}) \equiv \forall X.\, (\mathsf{InInitSet}(X, s, \mathcal{K}) \supset \mathsf{Honest}(\hat{X}))$$

Our objective is to state that secrets established by *secretive protocols*, where possibly the secrets are also used as keys, are good keys against everybody except the set of people who either generated the secret or are in possession of a key protecting the secret. The formula $\mathsf{GoodKeyFor}$ expresses this property. For level-0 keys that we want to claim are possessed only by honest principals we use the formula $\mathsf{GoodKey}$.

$$\begin{aligned}\mathsf{GoodKeyFor}(s,\mathcal{K}) &\equiv \forall X.\, (\mathsf{GoodKeyAgainst}(X,s) \vee \mathsf{InInitSet}(X,s,\mathcal{K}))\\ \mathsf{GoodKey}(k) &\equiv \forall X.\, (\mathsf{Possess}(X,k) \supset \mathsf{Honest}(\hat{X}))\end{aligned}$$

For protocols employing an IND-CCA secure encryption scheme, the soundness of the following axiom is based on Theorems 2 and 5:

$$\textbf{GK} \quad \mathsf{Secretive}(s,\mathcal{K}) \wedge \mathsf{GoodInit}(s,\mathcal{K}) \Rightarrow \mathsf{GoodKeyFor}(s,\mathcal{K})$$

If the encryption scheme is both IND-CCA and INT-CTXT secure then, the soundness of the following axioms are based on Theorems 3 and 6:

$$\begin{aligned}\textbf{CTX0} \quad & \mathsf{GoodKey}(k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \Rightarrow \exists X.\, \mathsf{SymEnc}(X,m,k),\\ & \quad \text{for level-0 key } k.\\ \textbf{CTXL} \quad & \mathsf{Secretive}(s,\mathcal{K}) \wedge \mathsf{GoodInit}(s,\mathcal{K}) \wedge \mathsf{SymDec}(Z, E_{sym}[s](m), s)\\ & \quad \Rightarrow \exists X.\, \mathsf{SymEnc}(X,m,s)\end{aligned}$$

The **soundness theorem** is proved by showing that every axiom is a valid formula and that all proof rules preserve validity.

Generalized Induction Principle. We present a generalized induction rule **IND** below. Two instances of this rule—$\textbf{IND}_{GOOD}$, described in the next subsection, and the **HON** rule, developed in prior work (see section 3.1.4)—are used to establish invariants in secrecy and authentication proofs respectively. The main idea is that if all honest threads executing some basic sequence (or protocol step) in the protocol locally preserve some property $\psi(\cdot)$, then we can conclude that $\psi(\cdot)$ holds in all states for all honest agents. The rule is strengthened with the formula Φ which lets us plug in any required assumptions. Note that a set of basic sequences (BS) of a role ρ is any partition of the sequence of actions in a role such that if any sequence has a `receive` then it is only at its beginning. This rule schema depends on the protocol. In the syntactic presentation below, we use the notation $\forall \rho \in \mathcal{Q}. \forall P \in BS(\rho). \psi(X)\, [P]_X \, \Phi \supset$

$\mathsf{SendGood}(X, s, \mathcal{K})$	$\equiv \forall m.\ (\mathsf{Send}(X, m) \supset \mathsf{Good}(X, m, s, \mathcal{K}))$
$\mathsf{Secretive}(s, \mathcal{K})$	$\equiv \forall X.\ (\mathsf{Honest}(\hat{X}) \supset \mathsf{SendGood}(X, s, \mathcal{K}))$
$\mathsf{InInitSet}(X, s, \mathcal{K})$	$\equiv \exists k \in \mathcal{B}(\mathcal{K}).\ \mathsf{Possess}(X, k) \vee \mathsf{New}(X, s)$
$\mathsf{GoodInit}(s, \mathcal{K})$	$\equiv \forall X.\ (\mathsf{InInitSet}(X, s, \mathcal{K}) \supset \mathsf{Honest}(\hat{X}))$
$\mathsf{GoodKeyFor}(s, \mathcal{K})$	$\equiv \forall X.\ (\mathsf{GoodKeyAgainst}(X, s) \vee \mathsf{InInitSet}(X, s, \mathcal{K}))$
$\mathsf{GoodKey}(k)$	$\equiv \forall X.\ (\mathsf{Possess}(X, k) \supset \mathsf{Honest}(\hat{X}))$

Table 3.4: Summary of New Definitions

$\psi(X)$ to denote a finite set of formulas of the form $\psi(X)\ [P]_X\ \Phi \supset \psi(X)$ - one for each basic sequence P in the protocol. For example, the first stage of Kerberos (as described in section 3.3.1) has three basic sequences - two of the client and one of the KAS. So there are three formulas for the first stage. In addition, there are six more from the other stages.

$$\textbf{IND} \quad \frac{\mathsf{Start}(X)\ [\,]_X\ \psi(X) \qquad \forall \rho \in \mathcal{Q}.\ \forall P \in BS(\rho).\ \psi(X)\ [P]_X\ \Phi \supset \psi(X)}{\mathcal{Q} \vdash \Phi \supset \forall X'.\ (\mathsf{Honest}(\hat{X'}) \supset \psi(X'))}(*)$$

(*): Φ is a prefix closed trace property which shares no free variable with $[P]_X$ and $\psi(X)$; $\psi(X)$ is a trace property which shares no free variable with $[P]_X$ except X.

A *prefix closed* formula Φ is a formula such that if a protocol trace t satisfies Φ then any prefix of t also satisfies Φ. For example, the formula $\neg\mathsf{Send}(X, m)$ is prefix closed. This is because if in any trace t, thread X has not sent the term m, it cannot have sent m in any prefix of t. In general, the negation of any action formula is prefix closed. Another example is $\forall X. \mathsf{New}(X, s) \supset \hat{X} = \hat{A}$ because this can be re-written as $\forall X.\ \neg\mathsf{New}(X, s) \vee \hat{X} = \hat{A}$ which is a disjunction of the negation of an action formula and an equality constraint.

The new definitions in this chapter are summarized in Table 3.4 and the new axioms and proof rules are summarized in Table 3.5.

Theorem 9 (Soundness) $\forall \mathcal{Q}, \varphi.$ *if* $\mathcal{Q} \vdash \varphi$ *then* $\mathcal{Q} \models \varphi$

G0	$\mathsf{Good}(X, a, s, \mathcal{K})$, if a is of an atomic type different from nonce or key
G1	$\mathsf{New}(Y, n) \wedge n \neq s \supset \mathsf{Good}(X, n, s, \mathcal{K})$
G2	$[\texttt{receive}\ m;]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$
G3	$\mathsf{Good}(X, m, s, \mathcal{K})\ [\mathtt{a}]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$, for all actions a
G4	$\mathsf{Good}(X, m, s, \mathcal{K})\ [\texttt{match}\ m\ \texttt{as}\ m';]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$
G5	$\mathsf{Good}(X, m_0, s, \mathcal{K}) \wedge \mathsf{Good}(X, m_1, s, \mathcal{K})\ [m := m_0.m_1;]_X\ \mathsf{Good}(X, m, s, \mathcal{K})$
G6	$\mathsf{Good}(X, m, s, \mathcal{K})\ [\texttt{match}\ m\ \texttt{as}\ m_0.m_1;]_X\ \mathsf{Good}(X, m_0, s, \mathcal{K}) \wedge \mathsf{Good}(X, m_1, s, \mathcal{K})$
G7	$\mathsf{Good}(X, m, s, \mathcal{K}) \vee k \in \mathcal{K}\ [m' := \texttt{symenc}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$
G8	$\mathsf{Good}(X, m, s, \mathcal{K}) \wedge k \notin \mathcal{K}\ [m' := \texttt{symdec}\ m, k;]_X\ \mathsf{Good}(X, m', s, \mathcal{K})$

SG0	$\mathsf{Start}(X)\ [\,]_X\ \mathsf{SendGood}(X, s, \mathcal{K})$
SG1	$\mathsf{SendGood}(X, s, \mathcal{K})\ [\mathtt{a}]_X\ \mathsf{SendGood}(X, s, \mathcal{K})$, where $\mathtt{a}$ is not a send.
SG2	$\mathsf{SendGood}(X, s, \mathcal{K})\ [\texttt{send}\ m;]_X\ \mathsf{Good}(X, m, s, \mathcal{K}) \supset \mathsf{SendGood}(X, s, \mathcal{K})$

IND$_{GOOD}$ $\forall \rho \in \mathcal{Q}. \forall P \in BS(\rho).$

$$\frac{\mathsf{SendGood}(X, s, \mathcal{K})\ [P]_X\ \Phi \supset \mathsf{SendGood}(X, s, \mathcal{K})}{\mathcal{Q} \vdash \Phi \supset \mathsf{Secretive}(s, \mathcal{K})}\ (*)$$

$(*)$: $[P]_X$ does not capture free variables in Φ, $\mathcal{K}$, s, and Φ is a prefix closed trace formula.

If the encryption scheme is IND-CCA secure then:

GK $\mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \Rightarrow \mathsf{GoodKeyFor}(s, \mathcal{K})$

If the encryption scheme is both IND-CCA and INT-CTXT secure then:

CTX0	$\mathsf{GoodKey}(k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \Rightarrow \exists X.\ \mathsf{SymEnc}(X, m, k)$, for level-0 key k.
CTXL	$\mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \wedge \mathsf{SymDec}(Z, E_{sym}[s](m), s) \Rightarrow \exists X.\ \mathsf{SymEnc}(X, m, s)$

Table 3.5: Summary of New Axioms and Proof Rules

Proof. Since the depth of any proof tree is constant with respect to the security parameter, it is sufficient to prove that each axiom and inference rule is sound.

IND : Let us denote $\Psi \equiv \forall X'.(\mathsf{Honest}(\hat{X}') \supset \psi(X'))$. Suppose there is an algorithm $\mathcal{A}$ which breaks $\Phi \supset \Psi$. That is, the set $T' = [\![\Phi]\!]\, T \cap [\![\neg\Psi]\!]\, T$ is a non-negligible subset of T, where $T = T_{\mathcal{Q}}(\mathcal{A}, \eta)$. We are going to assume the first premise to be valid and show that the second premise is invalid.

Let $T_{st} = [\![\forall X.\, \mathsf{Start}(X)\, [\,]_X\, \psi(X)]\!]\, T$. Following the first premise, we know that T_{st} is an overwhelming subset of T. Therefore the set $\tilde{T} = T' \cap T_{st}$ is a non-negligible subset of T. Consider a trace $t \in \tilde{T}$: following the construction of the set $\tilde{T}$, we have $\neg\psi(Y)$ true at the end of t for some thread Y, but $\psi(Y)$ true at any prefix of t where Y has not started. Due to this, there has to be a basic sequence P executed by Y such that $\psi(Y)$ is true before the sequence but false after it. Since there are a polynomial number of (basic sequence, thread) pairs, there must be a pattern $[\pi]_\chi$ which has this property in a polynomial fraction of $\tilde{T}$ - and hence is a non-negligible subset of T. Let us call this set $\tilde{T_\pi}$.

We are going to show that the second premise of the rule is invalid by showing that there is an algorithm $\mathcal{A}'$ such that the complement of the semantics of $\psi(\chi)\, [\pi]_\chi\, \Phi \supset \psi(\chi)$ is a non-negligible subset of its set of traces. The algorithm $\mathcal{A}'$ simulates $\mathcal{A}$ to the very end. It then scans the trace for the pattern $[\pi]_\chi$ such that $\psi(\chi)$ holds before the pattern and $\neg\psi(\chi)$ after it - this can be done in polynomial time. If such a condition occurs then it outputs markers corresponding to $[\pi]_\chi$ else outputs random markers. Note that $\mathcal{A}'$ produces exactly the same set of traces as $\mathcal{A}$ does - *i.e.*, $T_{\mathcal{Q}}(\mathcal{A}', \eta) = T_{\mathcal{Q}}(\mathcal{A}, \eta) = T$.

The size of the complement of $[\![\psi(\chi)\ [\pi]_\chi\ \Phi \supset \psi(\chi)]\!]\ T_{\mathcal{Q}}(\mathcal{A}', \eta)$ is:

$$\begin{aligned}
& |T| - |T_{\neg\pi} \cup [\![\neg\psi(\chi)]\!]\ \mathit{Pre}(T_\pi) \cup [\![\Phi \supset \psi(\chi)]\!]\ \mathit{Post}(T_\pi)| \\
&= |T| - |T_{\neg\pi} \cup [\![\neg\psi(\chi)]\!]\ \mathit{Pre}(T_\pi) \cup [\![\neg\Phi]\!]\ \mathit{Post}(T_\pi) \cup [\![\psi(\chi)]\!]\ \mathit{Post}(T_\pi)| \\
&\geq |T_\pi \cap [\![\psi(\chi)]\!]\ \mathit{Pre}(T_\pi) \cap [\![\neg\psi(\chi)]\!]\ \mathit{Post}(T_\pi) \cap [\![\Phi]\!]\ \mathit{Post}(T_\pi)| \\
&\geq \left|T_\pi \cap \tilde{T}_\pi \cap [\![\Phi]\!]\ \mathit{Post}(T_\pi)\right| \geq \left|\tilde{T}_\pi \cap [\![\Phi]\!]\ \mathit{Post}(T_\pi)\right| \geq \left|\tilde{T}_\pi \cap [\![\Phi]\!]\ \mathit{Post}(\tilde{T}_\pi)\right| \\
&= \left|\tilde{T}_\pi\right|
\end{aligned}$$

which is a non-negligible fraction of $|T|$ - hence the proof.

G0 – G8 : These axioms readily follow from the semantics of the predicate Good.

IND$_{GOOD}$: This is a specific instance of the rule **IND** obtained by putting $\psi(X) \equiv \mathsf{SendGood}(X, s, \mathcal{K})$.

SG0 – SG2 : These axioms readily follow from the definition of SendGood.

GK : For level-0 keys $\mathcal{K}$ the axiom will be valid if in all the traces where s is protected with keys $\mathcal{K}$ by a secretive protocol and where all the keys in $\mathcal{K}$ are IND-CCA secure against dishonest parties, no probabilistic poly-time distinguisher, given the view of a principal other than the key-holders and the generator of s has non-negligible advantage against an IND-CCA challenger using the bitstring corresponding to k as the key.

For set of keys $\mathcal{K}$ of level ≤ 1, the axiom will be valid if in all the traces where s is protected with keys $\mathcal{K}$ by a secretive protocol and where all the level-0 keys in $\mathcal{K}$ are IND-CCA secure against dishonest parties and all the level-1 keys are protected by level-0 keys which are again IND-CCA secure against dishonest parties, no probabilistic poly-time distinguisher, given the view of a principal other than the holders of keys in $\mathcal{C}(\mathcal{K})$ and the generator of s has non-negligible advantage against an IND-CCA challenger using the bitstring corresponding to k as the key.

The validity of these statements follows from Theorems 2 and 5.

CTX0 : For level-0 key k possessed only by honest principals, the axiom will be valid if in an overwhelmingly large subset of all the traces where an honest principal decrypts a message with key k, there was an honest principal who produced the message by encrypting with k. The validity of this message follows straightforwardly from the INT-CTXT [14] security of the encryption scheme.

CTXL : Here we consider symmetric encryption schemes that are both IND-CCA and INT-CTXT secure. For level-0 keys $\mathcal{K}$ held by honest principals and nonce s protected with keys $\mathcal{K}$ by a secretive protocol the axiom will be valid if in an overwhelmingly large subset of all the traces where an honest principal decrypts a message with key s, there was an honest principal who produced the message by encrypting with s. Analogously for key DAGs. The validity of this follows from Theorems 3 and 6. $\square$

3.2.3 Compositional Reasoning

In this section, we present composition theorems that allow *secretive*-ness proofs of compound protocols to be built up from proofs of their parts. We consider three kinds of composition operations on protocols—*parallel*, *sequential*, and *staged*—all based on the previous work by [33, 46]. However, adapting that approach for reasoning about secrecy requires new insights. One central concept in the compositional proof methods is the notion of an *invariant*. An invariant for a protocol is a logical formula that characterizes the environment in which it retains its security properties. While in [33] there is the honesty rule **HON** for establishing invariants, reasoning about *secretive*-ness requires a more general form of induction, captured in this paper by the **IND** rule. In addition, to proving that a protocol step does not violate *secretive*-ness, we need to employ derivations from earlier steps executed by the principal. In the technical presentation, this history information shows up as preconditions in the secrecy induction of the sequential and staged composition theorems. Instead of stating the theorems in terms of the **IND** rule, we keep the focus on its specific instance $\mathbf{IND}_{GOOD}$ to keep the presentation more concretely geared towards secrecy.

Definition 23 (Parallel Composition) *The parallel composition $\mathcal{Q}_1 \,|\, \mathcal{Q}_2$ of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is the union of the sets of roles of $\mathcal{Q}_1$ and $\mathcal{Q}_2$.*

The parallel composition operation allows modelling principals who simultaneously engage in sessions of multiple protocols. The parallel composition theorem provides a method for ensuring that security properties established independently for the constituent protocols are still preserved in such a situation.

Theorem 10 (Parallel Composition) *If $\mathcal{Q}_1 \vdash \Gamma$ and $\Gamma \vdash \Psi$ and $\mathcal{Q}_2 \vdash \Gamma$ then $\mathcal{Q}_1 \,|\, \mathcal{Q}_2 \vdash \Psi$, where Γ denotes the set of invariants used in the proof of Ψ.*

Definition 24 (Sequential Composition) *A protocol $\mathcal{Q}$ is a sequential composition of two protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$, if each role of $\mathcal{Q}$ is obtained by the sequential composition of a role of $\mathcal{Q}_1$ with a role of $\mathcal{Q}_2$.*

In practice, key exchange is usually followed by a secure message transmission protocol which uses the resulting shared key to protect data. Sequential composition is useful to model such compound protocols. Formally, the composed role $P_1; P_2$ is obtained by concatenating the actions of P_1 and P_2 with the output parameters of P_1 substituted for the input parameters of P_2 (cf. [33]).

Theorem 11 (Sequential Composition) *If $\mathcal{Q}$ is a sequential composition of protocols $\mathcal{Q}_1$ and $\mathcal{Q}_2$ then we can conclude $\mathcal{Q} \vdash \Phi \supset \mathsf{Secretive}(s, \mathcal{K})$ if the following conditions hold for all $P_1; P_2$ in $\mathcal{Q}$, where $P_1 \in \mathcal{Q}_1$ and $P_2 \in \mathcal{Q}_2$:*

1. *(Secrecy induction)*
   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i} \wedge \mathsf{SendGood}(X, s, \mathcal{K})\, [S]_X\, \Phi \supset \mathsf{SendGood}(X, s, \mathcal{K})$
2. *(Precondition induction)*
   - $\mathcal{Q}_1 \,|\, \mathcal{Q}_2 \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \,|\, \mathcal{Q}_2 \vdash \theta_{P_1} [P_1]_X\, \theta_{P_2}$
   - $\forall i. \forall S \in BS(P_i).\, \theta_{P_i} [S]_X\, \theta_{P_i}$.

The final conclusion of the theorem is a statement that the composed protocol is secretive with respect to s and $\mathcal{K}$. The secrecy induction is similar to the $\mathbf{IND}_{GOOD}$ rule. It states that all basic sequences of the two roles only send out good messages. This step is compositional since the condition is proved independently for steps of the two protocols. One point of difference from the $\mathbf{IND}_{GOOD}$ rule is the additional precondition θ_{P_i}. This formula usually carries some information about the history of the execution, which helps in deciding what messages are good for X to send out. For example, if θ_{P_i} says that X has previously received the message m, then it is easy to establish that m is a good message for X to send out again. The precondition induction requires that the θ_{P_i}'s hold at each point where they are required in the secrecy induction. The first bullet states the base case of the induction: θ_{P_1} holds at the beginning of the execution and θ_{P_2} holds when P_1 completes. The second bullet states that the basic sequences of P_1 and P_2 preserve their respective preconditions.

Definition 25 (Staged Composition) *A protocol $\mathcal{Q}$ is a staged composition of protocols $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ if each role of $\mathcal{Q}$ is in $RComp(\langle R_1, R_2 ... R_n\rangle)$, where R_i is a role of protocol $\mathcal{Q}_i$.*

Consider the representation of sequential composition of n protocols as a directed graph with edges from $\mathcal{Q}_i$ to $\mathcal{Q}_{i+1}$. The staged composition operation extends sequential composition by allowing self loops and arbitrary backward arcs in this chain. This control flow structure is common in practice, e.g., Kerberos [52], IEEE 802.11i [1], and IKEv2 [23], with backward arcs usually corresponding to error handling or rekeying. A role in this composition, denoted $RComp(\langle ... \rangle)$ corresponds to a possible execution path in the control flow graph by a single thread (cf. [46]). Note that the roles are built up from a finite number of basic sequences of the component protocol roles.

Theorem 12 (Staged Composition) *If $\mathcal{Q}$ is a staged composition of protocols $\mathcal{Q}_1$, $\mathcal{Q}_2$, $\cdots$, $\mathcal{Q}_n$, then we can conclude $\mathcal{Q} \vdash \Phi \supset \mathsf{Secretive}(s, \mathcal{K})$ if for all $RComp(\langle P_1, P_2, \cdots , P_n\rangle) \in \mathcal{Q}$:*

1. *(Secrecy induction)*

   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i} \wedge \mathsf{SendGood}(X, s, \mathcal{K})\ [S]_X\ \Phi \supset \mathsf{SendGood}(X, s, \mathcal{K})$

2. *(Precondition induction)*

   - $\mathcal{Q}_1 \mid \mathcal{Q}_2 \cdots \mid \mathcal{Q}_n \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \mid \mathcal{Q}_2 \cdots \mid \mathcal{Q}_n \vdash \forall i.\, \theta_{P_i}[P_i]_X\, \theta_{P_{i+1}}$
   - $\forall i. \forall S \in \bigcup_{j \geq i} BS(P_j).\, \theta_{P_i}[S]_X\, \theta_{P_i}$.

The secrecy induction for staged composition is the same as for sequential composition. However, the precondition induction requires additional conditions to account for the control flows corresponding to backward arcs in the graph. The technical distinction surfaces in the second bullet of the precondition induction. It states that precondition θ_{P_i} should also be preserved by basic sequences of all higher numbered components, i.e., components from which there could be backward arcs to the beginning of P_i.

3.3 Kerberos V5

We illustrate the proof system using Kerberos V5 [52]. Our formulation is based on the A level formalization of Kerberos V5 in [22]. Kerberos provides mutual authentication and establishes keys between clients and application servers, using a sequence of two-message interactions with trusted parties called the Kerberos Authentication Server (KAS) and the Ticket Granting Server (TGS).

3.3.1 Modeling Kerberos

Protocols are expressed in a process calculus by defining a set of *roles*, such as "Client", or "Server", each given by a sequence of actions such as sending or receiving a message, generating a new nonce, or decrypting or encrypting a message (see [33]). In a run of a protocol, a principal may execute one or more instances of each role, each execution constituting a *thread* identified by a pair $(\hat{X}, \eta)$, where $\hat{X}$ is a principal and η is a unique session identifier.

Kerberos has four roles, **Client**, **KAS**, **TGS** and **Server**. The pre-shared long-term keys between the client and KAS, the KAS and TGS, and the TGS and application server, will be written as $k_{X,Y}^{type}$ where X and Y are the principals sharing the key. The *type* appearing in the superscript indicates the relationship between X and Y: $c \rightarrow k$ indicates that X is acting as a client and Y is acting as a KAS, $t \rightarrow k$ for TGS and KAS and $s \rightarrow t$ for application server and TGS. The formal description is given in Table 3.6.

In the first stage, the client (C) generates a nonce (represented by `new` n_1) and sends it to the KAS (K) along with the identities of the TGS (T) and itself. The KAS generates a new nonce ($AKey$ - Authentication Key) to be used as a session key between the client and the TGS. It then sends this key along with some other fields to the client encrypted (represented by the `symenc` actions) under two different keys - one it shares with the client ($k^{c\rightarrow k}_{C,K}$) and one it shares with the TGS ($k^{t\rightarrow k}_{T,K}$). The encryption with $k^{t\rightarrow k}_{T,K}$ is called the ticket granting ticket (tgt). The client extracts $AKey$ by decrypting the component encrypted with $k^{c\rightarrow k}_{C,K}$ and recovering its parts using the `match` action which deconstructs $text_{kc}$ and associates the parts of this plaintext with $AKey$, n_1, and $\hat{T}$. The ellipses $(\ldots)$ indicates further client steps for interacting with KAS, TGS, and the application server that are omitted due to space constraints (see [68] for a full description).

In the second stage, the client gets a new session key ($SKey$ - Service Key) and a service ticket (st) to converse with the application server S which takes place in the third stage. The control flow of Kerberos exhibits a staged architecture where once one stage has been completed successfully, the subsequent stages can be performed multiple times or aborted and started over for handling errors.

3.3.2 Analysis of Kerberos

Table 3.7 lists the security properties of Kerberos that we prove. The security objectives are of two types: authentication and secrecy. The authentication objectives take the form that a message of a certain format was indeed sent by some thread of the expected principal. The secrecy objectives take the form that a putative secret is a good key for certain principals. For example, $AUTH^{client}_{kas}$ states that when C finishes executing the **Client** role, some thread of $\hat{K}$ indeed sent the expected message; SEC^{client}_{akey} states that the authorization key is good after execution of the **Client** role

```
**Client** $= (C, \hat{K}, \hat{T}, \hat{S}, t)$ [
  new $n_1$;
  send $\hat{C}.\hat{T}.n_1$;

  receive $\hat{C}.tgt.enc_{kc}$;
  $text_{kc}$ := symdec $enc_{kc}, k^{c\to k}_{C,K}$;
  match $text_{kc}$ as $AKey.n_1.\hat{T}$;


  $\cdots$ stage boundary $\cdots$

  new $n_2$;
  $enc_{ct}$ := symenc $\hat{C}, AKey$;
  send $tgt.enc_{ct}.\hat{C}.\hat{S}, n_2$;

  receive $\hat{C}.st.enc_{tc}$;
  $text_{tc}$ := symdec $enc_{tc}, AKey$;
  match $text_{tc}$ as $SKey.n_2.\hat{S}$;


  $\cdots$ stage boundary $\cdots$


  $enc_{cs}$ := symenc $\hat{C}.t, SKey$;
  send $st.enc_{cs}$;

  receive $enc_{sc}$;
  $text_{sc}$ := symdec $enc_{sc}, SKey$;
  match $text_{sc}$ as $t$;
  ]$_C$
```

```
**KAS** $= (K)$ [
  receive $\hat{C}.\hat{T}.n_1$;
  new $AKey$;
  $tgt$ := symenc $AKey.\hat{C}, k^{t\to k}_{T,K}$;
  $enc_{kc}$ := symenc $AKey.n_1.\hat{T}, k^{c\to k}_{C,K}$;
  send $\hat{C}.tgt.enc_{kc}$;
  ]$_K$

**TGS** $= (T, \hat{K})$ [
  receive $tgt.enc_{ct}.\hat{C}.\hat{S}.n_2$;
  $text_{tgt}$ := symdec $tgt, k^{t\to k}_{T,K}$;
  match $text_{tgt}$ as $AKey.\hat{C}$;
  $text_{ct}$ := symdec $enc_{ct}, AKey$;
  match $text_{ct}$ as $\hat{C}$;
  new $SKey$;
  $st$ := symenc $SKey.\hat{C}, k^{s\to t}_{S,T}$;
  $enc_{tc}$ := symenc $SKey.n_2.\hat{S}, AKey$;
  send $\hat{C}.st.enc_{tc}$;
  ]$_T$

**Server** $= (S, \hat{T})$ [
  receive $st.enc_{cs}$;
  $text_{st}$ := symdec $st, k^{s\to t}_{S,T}$;
  match $text_{st}$ as $SKey.\hat{C}$;
  $text_{cs}$ := symdec $enc_{cs}, SKey$;
  match $text_{cs}$ as $\hat{C}.t$;
  $enc_{sc}$ := symenc $t, SKey$;
  send $enc_{sc}$;
  ]$_S$
```

Table 3.6: Formal Description of Kerberos V5 Roles.

$$\begin{aligned}
SEC_{akey} &: \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset (\mathsf{GoodKeyAgainst}(X, AKey) \vee \hat{X} \in \{\hat{C},\hat{K},\hat{T}\})\\
SEC_{skey} &: \mathsf{Hon}(\hat{C},\hat{K},\hat{T},\hat{S}) \supset (\mathsf{GoodKeyAgainst}(X, SKey) \vee \hat{X} \in \{\hat{C},\hat{K},\hat{T},\hat{S}\})\\
AUTH_{kas} &: \exists \eta.\ \mathsf{Send}((\hat{K},\eta), \hat{C}.E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C}).E_{sym}[k^{c\to k}_{C,K}](AKey.n_1.\hat{T}))\\
AUTH_{tgs} &: \exists \eta.\ \mathsf{Send}((\hat{T},\eta), \hat{C}.E_{sym}[k^{s\to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))
\end{aligned}$$

$$\begin{array}{rlrl}
SEC^{client}_{akey} & : [\textbf{Client}]_C\ SEC_{akey} & AUTH^{client}_{kas} & : [\textbf{Client}]_C\ \mathsf{Hon}(\hat{C},\hat{K}) \supset AUTH_{kas}\\
SEC^{kas}_{akey} & : [\textbf{KAS}]_K\ SEC_{akey} & AUTH^{tgs}_{kas} & : [\textbf{TGS}]_T\ \mathsf{Hon}(\hat{T},\hat{K}) \supset \exists n_1.\ AUTH_{kas}\\
SEC^{tgs}_{akey} & : [\textbf{TGS}]_T\ SEC_{akey} & & \\
 & & AUTH^{client}_{tgs} & : [\textbf{Client}]_C\ \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset AUTH_{tgs}\\
SEC^{client}_{skey} & : [\textbf{Client}]_C\ SEC_{skey} & AUTH^{server}_{tgs} & : [\textbf{Server}]_S\ \mathsf{Hon}(\hat{S},\hat{T})\\
SEC^{tgs}_{skey} & : [\textbf{TGS}]_T\ SEC_{skey} & & \quad \supset \exists n_2, AKey.\ AUTH_{tgs}
\end{array}$$

Table 3.7: Kerberos Security Properties

by C; the other security properties are analogous. We abbreviate the honesty assumptions by defining $\mathsf{Hon}(\hat{X}_1, \hat{X}_2, \cdots, \hat{X}_n) \equiv \mathsf{Honest}(\hat{X}_1) \wedge \mathsf{Honest}(\hat{X}_2) \wedge \cdots \mathsf{Honest}(\hat{X}_n)$. The formal proofs using the proof system are given in Appendix A. In this section we provide informal explanations.

The overall proof structure demonstrates an interleaving of authentication and secrecy properties, reflecting the intuition behind the protocol design. We start with proving some authentication properties based on the presumed secrecy of long-term shared symmetric keys. As intended in the design, these authentication guarantees enable us to prove the secrecy of data protected by the long-term keys. This general theme recurs further down the protocol stages. Part of the data is used in subsequent stages as an encryption key. The secrecy of this transmitted encryption key lets us establish authentication in the second stage of the protocol. The transmitted key is also used to protect key exchange in this stage - the secrecy of which depends on the authentication established in the stage.

Theorem 13 (KAS Authentication) *On execution of the* **Client** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended KAS indeed sent the expected response assuming that both the client and the KAS are honest. A similar result holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash AUTH_{kas}^{client}, AUTH_{kas}^{tgs}$.

Authentication is achieved by the virtue of ciphertext integrity offered by the symmetric encryption scheme. At a high level, we reason that a ciphertext could have been produced only by one of the possessors of the corresponding key.

As an example, observe that in the first stage of Kerberos (described in Section 3.1), the client decrypts a ciphertext encrypted with a key shared only between itself and the KAS ($k_{C,K}^{c\to k}$). Hence it should be overwhelmingly probable that one of them did the encryption. This reasoning is formally captured by the axiom **CTX0**. However, it is still not obvious that the client itself did not produce the ciphertext! Some other thread of the client could have potentially created the ciphertext which could have been fed back to the thread under consideration as a reflection attack. We discount this case by observing that the client role of Kerberos never encrypts with a key of type $c \to k$. This property is an *invariant* of Kerberos proved by induction over all the protocol role programs. The **HON** rule enables us to perform this induction in the proof system. Thus, so far, we have reasoned that the encryption was done by the KAS. We again observe that any role of Kerberos which does an encryption of the specific form as in stage one also sends out a message of the intended form ($AUTH_{kas}$ in Table 3.7). This is also an invariant of Kerberos. We now have a high level intuition of the entire proof.

The formal proof follows this high level intuition. In course of execution of the **Client** role by principal $\hat{C}$, it decrypts the message $E_{sym}[k_{C,K}^{c\to k}](AKey.n_1.\hat{T})$. Using axiom **CTX0**, we derive that it was encrypted by one of the owners of $k_{C,K}^{c\to k}$ - i.e. either $\hat{C}$, or $\hat{K}$. Then, by using the invariant rule **HON**, we establish that no thread of $\hat{C}$ does this (assuming $\hat{C} \neq \hat{K}$) - hence it must be some thread of $\hat{K}$ (also, this trivially holds if $\hat{C} = \hat{K}$). Once again we use the **HON** rule to reason that if an honest thread encrypts a message of this form then it also sends out a message of the form described in $AUTH_{kas}$. The proof of $AUTH_{kas}^{tgs}$ is along identical lines. In the formal proof, we first give a *template* proof for the underlying reasoning and then instantiate it for both $AUTH_{kas}^{client}$ and $AUTH_{kas}^{tgs}$. In $AUTH_{kas}^{tgs}$, the existential quatification over n_1 is there because T is oblivious to what n_1 was used in the interaction between $\hat{C}$ and $\hat{K}$ but it can still infer that *some* n_1 was used.

Theorem 14 (Authentication Key Secrecy) *On execution of the* **Client** *role by a principal, the Authentication Key is guaranteed to be good, in the sense of IND-CCA security, assuming that the client, the KAS and the TGS are all honest. Similar results hold for principals executing the* **KAS** *and* **TGS** *roles. Formally,* $KERBEROS \vdash SEC_{akey}^{client}, SEC_{akey}^{kas}, SEC_{akey}^{tgs}$.

Proof Sketch. This theorem states the secrecy property for the Authentication Key $AKey$ - that $AKey$ satisfies IND-CCA key usability. Observe that in the first stage, the KAS sends out $AKey$ encrypted under two different keys - $k_{C,K}^{c\to k}$ and $k_{T,K}^{t\to k}$, and the client uses $AKey$ as an encryption key. As a first approximation we conjecture that in the entire protocol execution, $AKey$ is either protected by encryption with either of the keys in $\mathcal{K} = \{k_{C,K}^{c\to k}, k_{T,K}^{t\to k}\}$ or else used as an encryption key in messages sent to the network by honest principals. This seems like a claim to be established by induction. As a base case, we establish that the generator of $AKey$ (some thread of the KAS) satisfies the conjecture. The induction case is: whenever an honest principal decrypts a ciphertext with one of the keys in $\mathcal{K}$, it ensures that new terms generated from the decryption are re-encrypted with some key in $\mathcal{K}$ in any message sent out. The results (of the appropriate type) from such a decryption are however, allowed to be used as encryption keys, which as you can note is the case in the first stage of the client.

When we are reasoning from the point of view of the KAS (as in SEC_{akey}^{kas}), we already know the initial condition - that the KAS sent out $AKey$ encrypted under only these keys. However, when arguing from the point of view of the client and the TGS (as in SEC_{akey}^{client} and SEC_{akey}^{tgs}), we need to have some authentication conditions established first. These conditions are generally of the form that the KAS indeed behaved in the expected manner. Reasoning from this premise, we prove that our initial conjecture is correct.

In the formal proof, we show that Kerberos is a *secretive protocol* with respect to the nonce $AKey$ and the set of keys $\mathcal{K}$. The induction idea is captured, in its simplest form, by the proof rule IND_{GOOD}. However, as Kerberos has a staged structure we use the staged composition theorem which builds upon the rule IND_{GOOD}. The core of the proof is the *secrecy induction* which is an induction over all the basic sequences of all the protocol roles. The authentication condition Φ is easily derived from the KAS Authentication theorem (Theorem 13). The staged composition theorem allows us to facilitate the secrecy induction by obtaining inferences from the information flow induced by the staged structure of Kerberos in a simple and effective way. The secrecy induction is modular as the individual basic sequences are small in themselves. Goodness of $AKey$ now follows from Theorem 5 (CCA security - level 1), which is formally expressed by axiom **GK**. A fragment of the proof is given in Table 3.8 - the full proof is in Appendix A.1.2.

Theorem 15 (TGS Authentication) *On execution of the* **Client** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended TGS indeed sent the expected response assuming that the client, the KAS and the TGS are all honest. A similar result holds for a principal executing the* **Server** *role. Formally, KERBEROS* $\vdash AUTH^{client}_{tgs}, AUTH^{server}_{tgs}$.

Theorem 16 (Service Key Secrecy) *On execution of the* **Client** *role by a principal, the Service Key is guaranteed to be good, in the sense of IND-CCA security, assuming that the client, the KAS, the TGS and the application server are all honest. A similar result holds for a principal executing the* **TGS** *role. Formally, KERBEROS* $\vdash SEC^{client}_{skey}, SEC^{tgs}_{skey}$.

The proof of $AUTH^{server}_{tgs}$ is similar to the proof of Theorem 13. The proof of $AUTH^{client}_{tgs}$ depends on the 'goodkey'-ness of $AKey$ established by Theorem 14. For Theorem 16, the idea is that the Service Key $SKey$ is protected by level-0 key $k^{s\to t}_{S,T}$ and level-1 key $AKey$. The proof of 'Secretive'-ness proceeds along the same line as for Theorem 14 and uses derivations from Theorem 15. Then we invoke axiom **GK** to establish *KERBEROS* $\vdash SEC^{client}_{skey}, SEC^{tgs}_{skey}$.

We formally prove the secrecy of the session key $AKey$ with respect to the set of keys $\mathcal{K} = \{k^{c\to k}_{C,K}, k^{t\to k}_{T,K}\}$. The assumed condition Φ is the conjunction of the following formulae:

$$\begin{aligned}
&\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))\\
&\Phi_2 : \forall X, \hat{C}_0, \hat{K}_0, \hat{T}_0, n.\ \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.n.\hat{T}_0, k^{c\to k}_{C_0,K_0})\\
&\qquad \supset \hat{X} = \hat{K} \wedge \hat{C}_0 = \hat{C} \wedge \hat{T}_0 = \hat{T}\\
&\Phi_3 : \forall X, \hat{S}_0, \hat{C}_0.\ \mathsf{New}(X, AKey) \supset \neg\mathsf{SymEnc}(X, AKey.\hat{C}_0, k^{s\to t}_{S_0,X})
\end{aligned}$$

Observe that Φ is prefix closed. Now we present the formal proof for the first three basic sequences of the **Client** role:

Let, $[\mathbf{Client}_1]_{C'} : [\texttt{new}\ n'_1; \texttt{send}\ \hat{C}'.\hat{T}'.n'_1;]_{C'}$

$$\begin{array}{rll}
 & [\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n'_1) \wedge \mathsf{Send}(C', \hat{C}'.\hat{T}'.n'_1) & (3.1)\\
\Phi_1, (-1) & [\mathbf{Client}_1]_{C'}\ n'_1 \neq AKey & (3.2)\\
\mathbf{G1}, \mathbf{G5}, (-1) & [\mathbf{Client}_1]_{C'}\ \mathsf{Good}(C', \hat{C}'.\hat{T}'.n'_1, AKey, \mathcal{K}) & (3.3)\\
\mathbf{SG1\text{-}2}, (-1) & \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) & (3.4)
\end{array}$$

Let, $[\mathbf{Client}_2]_{C'} : [\texttt{receive}\ \hat{C}'.tgt'.enc'_{kc};$
$text'_{kc} := \texttt{symdec}\ enc'_{kc}, k^{c\to k}_{C',K'};$
$\texttt{match}\ text'_{kc}\ \texttt{as}\ AKey'.n'_1.\hat{T}';]_{C'}$

$$\begin{array}{rll}
\mathbf{SG1} & \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) & (3.5)
\end{array}$$

Precondition $\theta_3 : \mathsf{Good}(C', tgt', AKey, \mathcal{K})$

Let, $[\mathbf{Client}_3]_{C'} : [\texttt{new}\ n'_2;\ enc'_{ct} := \texttt{symenc}\ \hat{C}', AKey';$
$\texttt{send}\ tgt'.enc'_{ct}.\hat{C}'.\hat{S}'.n'_2;]_{C'}$

$$\begin{array}{rll}
 & [\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n'_2) \wedge \mathsf{Send}(C', tgt'.enc'_{ct}.\hat{C}'.\hat{S}'.n'_2) & (3.6)\\
\Phi_1, (-1) & [\mathbf{Client}_3]_{C'}\ n'_2 \neq AKey & (3.7)\\
\mathbf{G1}, (-1) & \theta_3\ [\texttt{new}\ n'_2;]_{C'}\ \mathsf{Good}(C', tgt', AKey, \mathcal{K}) \wedge \mathsf{Good}(C', n'_2, AKey, \mathcal{K}) & (3.8)\\
\mathbf{G}*, (-1) & \theta_3\ [\mathbf{Client}_3]_{C'}\ \mathsf{Good}(C', tgt'.enc'_{ct}.\hat{C}'.\hat{S}'.n'_2, AKey, \mathcal{K}) & (3.9)\\
\mathbf{SG1\text{-}2}, (-1) & \theta_3 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) & (3.10)
\end{array}$$

Table 3.8: A fragment of the formal proof for secrecy of AKey.

Kerberos with PKINIT. In the first stage of Kerberos with PKINIT [71], the KAS establishes the authorization key encrypted with a symmetric key which in turn is sent to the client encrypted with its public key. Since the protocol uses both public and symmetric keys at level 0, we formulate a definition of a joint public-symmetric key game. We then extend the proof system and prove all the syntactically analogous properties of the PKINIT version.

For client $\hat{C}$ and KAS $\hat{K}$ let us denote this symmetric key by $k_{C,K}^{pkinit}$. Since the structure of the rest of the protocol remains the same with respect to the level of formalization in this chapter [27], we can take advantage of the CPCL proofs for the symmetric key version. In particular, the proofs for $AUTH_{kas}^{tgs}, AUTH_{tgs}^{client}$ and $AUTH_{tgs}^{server}$ proceed identically. The proof of $AUTH_{kas}^{client}$ is different because of the differing message formats in the first stage. There is an additional step of proving the secrecy of $k_{C,K}^{pkinit}$, after which the secrecy proofs of $AKey$ and $SKey$ are reused with only the induction over the first stage of the client and the KAS being redone.

The formal description of Kerberos PKINIT and security proofs are given in Appendix A.

Chapter 4

Proofs for Diffie-Hellman-based Protocols

Diffie-Hellman key exchange (DHKE) is one of the earliest public-key concepts [39]. It allows two parties without a prior shared secret to jointly create one that is independent of past and future keys, and is therefore widely used in many network security protocols. In this chapter, we develop axioms for reasoning about protocols that use Diffie-Hellman key exchange, prove these axioms sound using cryptographic reduction arguments, and use the axiom system to formally prove authentication and secrecy theorems for two significant standardized protocols. The two protocols we consider are Diffie-Hellman Key Exchange for initial authentication in Kerberos V5 [71] (which we refer to as DHINIT) and IKEv2 [23], the IPSEC key exchange standard. Kerberos is widely used in Microsoft Windows networking and other applications, while IKEv2 is part of IPSEC which is widely used for virtual private networks. The authentication and secrecy theorems, for probabilistic polynomial-time execution and standard cryptographic protocol attacks, have not been proved before to the best of our knowledge. In analyzing DHINIT, we also discover that the KAS is *not* authenticated to the client after the first stage, but we are able to prove formally in our logic that authentication is nonetheless achieved at a later stage; we also suggest a change to the protocol to ensure authentication after the first stage. In analyzing IKEv2, which replaces the controversial Internet Key Exchange (IKEv1) protocol using concepts from an intermediate protocol called Just Fast Keying (JFK) [4], we consider the IKEv2 mode in which signatures are used for authentication and Diffie-Hellman exponentials are never reused.

The axioms presented in this chapter are used in Protocol Composition Logic (PCL) [33, 35, 68, 34, 66]. Our formalization uses the characterization of "good key" from [36], but improves on previous work in several respects: (i) we fix a bug in the **DH** axiom in [36] by using the "DHStrongSecretive" formulas developed in this chapter, (ii) we present a general inductive method for proving secrecy conditions for Diffie-Hellman key exchange, and (iii) we present axioms for reasoning from ciphertext integrity assumptions. These three innovations are essential for the formal proofs for DHINIT and IKEv2, which could not be carried out in the system of [36]. In addition, the present soundness proofs are based on a new cryptographic definition and associated theorems about the joint security of multiple encryption schemes keyed using random or DHKE-keys. This chapter complements Chapter 3 and completes the development of formal cryptographically sound proofs for three modes of Kerberos V5.

Most demonstrated approaches for proving security of complex network protocols, of the scale that appear in IEEE and IETF standards, use a simplified model of protocol execution based on symbolic computation and highly idealized cryptography [12, 22, 27, 33]. However, proofs about symbolic computation do not provide the same level of assurance as proofs about probabilistic polynomial-time attacks. Several groups of researchers have therefore developed methods for deriving cryptographic meaning from properties of symbolic protocol execution [9, 5, 26, 30, 49, 50, 58]. These methods involve showing that the behavior of a symbolic abstraction, under symbolic attacks, yields the same significant failures as a finer-grained execution under finer-grained probabilistic polynomial-time attack. However, such equivalence theorems rely on strong cryptographic assumptions, and there are no known suitable symbolic abstractions of Diffie-Hellman exponentiation. In addition, there are theoretical negative results that suggest that correspondence theorems may be impossible for symmetric encryption if a protocol might reveal a secret key [25, 31], or for hash functions or exclusive-or [8, 10]. In contrast, computational PCL reasons directly about properties of probabilistic polynomial-time execution of protocols, under attack by a probabilistic polynomial-time adversary, without explicit formal reasoning about probability or complexity. In addition, different axioms depend on different cryptographic assumptions, allowing us to consider which assumptions are actually necessary for each property we establish. As currently formulated in the RFC, Kerberos requires a party to sign only its own Diffie-Hellman exponential. We prove this is sufficient, using axioms that depend on the *random oracle* assumption [15]. However, we are not able to give a formal proof using alternate axioms that do not depend on random oracles. On the other hand, the alternate axioms are sufficient to prove authentication if we modify the protocol slightly so that the KAS signs both the Diffie-Hellman exponentials, as is done in IKEv2 and JFK.

Two related studies are a symbolic proof for Kerberos (without DHKE) [6] and a cryptographic reduction proof for JFK [4]. In the Kerberos analysis, a correspondence between symbolic computation and cryptographic models [9] is used to draw cryptographic conclusions. This requires a separate verification that a "commitment problem" does not occur in the protocol (see [6]), and does not extend to Diffie-Hellman. The JFK proof is interesting and informative, with suggestions in [4] that "analysis based on formal methods would be a useful complement," but simpler than the proof of DHINIT since JFK digitally signs Diffie-Hellman values differently. More generally, Abadi and Rogaway [2] initiated computationally sound symbolic analysis of static equivalence, with extensions and completeness explored in [57, 3]; a recent extension to Diffie-Hellman appears in [21], covering only *passive adversaries,* not the stronger active adversaries used in this chapter. Protocol Composition Logic [33] was used in a case study of 802.11i [46], has previous computational semantics [35], and was used to study protocol composition and key exchange [36]. In other studies of DHKE, [48] uses a symbolic model, while [53] imposes nonstandard protocol assumptions. The cryptographic primitives used in Kerberos are analyzed in [20].

Section 4.1 presents the proof system and computational soundness theorem. Kerberos DHINIT and IKEv2 are analyzed in sections 4.2 and 4.3, respectively.

4.1 Proof System

Section 4.1.1 contains new axioms and rules for reasoning about Diffie-Hellman key exchange. Section 4.1.2 summarizes the concept of *secretive protocol* and proof rules taken from Chapter 3 that are used in this chapter to establish secrecy properties. However, we give new soundness proofs for these axioms, based on an extension of standard multiparty encryption schemes [13] to allow for multiple public and symmetric encryption schemes keyed using random or Diffie-Hellman based keys. The associated cryptographic definitions and theorems are presented in Section 4.1.3.

4.1.1 Diffie-Hellman Axioms

In this section we formalize reasoning about how individual threads treat DH exponentials in an appropriate way. We introduce the predicate $\mathsf{DHGood}(X, m, x)$, where x is a nonce used to compute a DH exponential, to capture the notion that thread X only uses certain safe actions to compute m from values that it has generated or received over the network. For example, axioms **DH2** and **DH3** say that a message m is DHGood if it has just been received, or if it is just computed by exponentiating the known group generator g with the nonce x. Axiom **DH4** states that the pair of two DHGood terms is also DHGood.

Semantics of DHGood: $[\![\mathsf{DHGood}(X, m, x)]\!](T, D, \epsilon)$ is the collection of all traces $t \in T$ where thread X constructs the term m in a 'good' way. Received messages, data of atomic type different from nonce or key, nonces different from x are all 'good' terms. Constructions that are 'good' consist of exponentiating g to the power x, pairing or unpairing good terms, encrypting and decrypting good terms and signing and hashing good terms.

Note that unlike the symbolic model, it is not well defined in the computational model to say "m contains x". That is why our proof systems for secrecy in the symbolic model [68] and computational model [66] are different - the computational system does induction on actions rather than structure of terms. The need to look at the structure of m is obviated by the way the reduction to games like IND-CCA works. The high level intuition is that a consistent simulation of the protocol can be performed while doing the reduction, if the terms to be sent to the adversary are "good".

DH0 $\mathsf{DHGood}(X, a, x)$, for a of any atomic type, except nonce, *viz.* name or key

DH1 $\mathsf{New}(Y, n) \wedge n \neq x \supset \mathsf{DHGood}(X, n, x)$

DH2 $[\mathtt{receive}\ m;]_X\ \mathsf{DHGood}(X, m, x)$

DH3 $[m := \mathtt{expg}\ x;]_X\ \mathsf{DHGood}(X, m, x)$

DH4 $\mathsf{DHGood}(X, m_0, x) \wedge \mathsf{DHGood}(X, m_1, x)\ [m := m_0.m_1;]_X\ \mathsf{DHGood}(X, m, x)$

DH5 $\mathsf{DHGood}(X, m, x)\ [m' := \mathtt{symenc}\ m, k;]_X\ \mathsf{DHGood}(X, m', x)$

DH6 $\mathsf{DHGood}(X, m, x)\ [m' := \mathtt{hash}\ m;]_X\ \mathsf{DHGood}(X, m', x)$

The formula $\mathsf{SendDHGood}(X, x)$ indicates that thread X sent out only "DHGood" terms w.r.t. the nonce x. $\mathsf{DHSecretive}$ (X, Y, k) means that there exist nonces x, y such that threads X, Y respectively generated them, sent out "DHGood" terms and X generated the key k from g^{xy}. $\mathsf{DHStrongSecretive}(X, Y, k)$ asserts a stronger condition - that threads X and Y only used each other's DH exponentials to generate the shared secret (The predicate $\mathsf{Exp}(X, gx, y)$ means thread X exponentiates gx to the power y). The formula $\mathsf{SharedSecret}(X, Y, k)$ means that the key k satisfies IND-CCA key usability against any thread other than X or Y, particularly against any adversary. Formally,

$$\begin{aligned}
&\mathsf{SendDHGood}(X, x) \equiv \forall m.\ \mathsf{Send}(X, m) \supset \mathsf{DHGood}(X, m, x)\\
&\mathsf{DHSecretive}(X, Y, k) \equiv \exists x, y.\ \mathsf{New}(X, x) \wedge \mathsf{SendDHGood}(X, x) \wedge\\
&\qquad \mathsf{New}(Y, y) \wedge \mathsf{SendDHGood}(Y, y) \wedge \mathsf{KeyGen}(X, k, x, g^y)\\
&\mathsf{DHStrongSecretive}(X, Y, k) \equiv \exists x, y.\ \mathsf{New}(X, x) \wedge \mathsf{SendDHGood}(X, x) \wedge\\
&\qquad \mathsf{New}(Y, y) \wedge \mathsf{SendDHGood}(Y, y) \wedge \mathsf{KeyGen}(X, k, x, g^y) \wedge\\
&\qquad (\mathsf{Exp}(X, gy, x) \supset gy = g^y) \wedge (\mathsf{Exp}(Y, gx, y) \supset gx = g^x)\\
&\mathsf{SharedSecret}(X, Y, k) \equiv \forall Z.\ \mathsf{GoodKeyAgainst}(Z, k) \vee Z = X \vee Z = Y
\end{aligned}$$

The following axioms hold for the above definition of SendGood:

SDH0 $\mathsf{Start}(X) \supset \mathsf{SendDHGood}(X, x)$

SDH1 $\mathsf{SendDHGood}(X, x)\ [\mathsf{a}]_X\ \mathsf{SendDHGood}(X, x)$, where a is not a send action

SDH2 $\mathsf{SendDHGood}(X, x)\ [\mathtt{send}\ m;]_X\ \mathsf{DHGood}(X, m, x) \supset \mathsf{SendDHGood}(X, x)$

The following axioms relate the DHStrongSecretive property, which is trace based, to computational notions of security. The first axiom, which depends on the DDH (Decisional Diffie-Hellman) assumption and IND-CCA security of the encryption scheme, states a secrecy property - if threads X and Y are DHStrongSecretive w.r.t. the key k, then k satisfies IND-CCA key usability. The second axiom, which depends on the DDH assumption and INT-CTXT (ciphertext integrity [14, 51]) security of the encryption scheme, states that with the same DHStrongSecretive property, if someone decrypts a term with the key k successfully, then it must have been encrypted with the key k by either X or Y. Both the axioms are proved sound by cryptographic reductions to the primitive security games.

DH $\mathsf{DHStrongSecretive}(X, Y, k) \Rightarrow \mathsf{SharedKey}(X, Y, k)$

CTXGS $\mathsf{DHStrongSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset$
$\mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)$

If the weaker property $\mathsf{DHSecretive}(X, Y, k)$ holds then we can establish an axiom similar to **CTXGS**, but we have to model the key generation function as a random oracle and the soundness proof is very different. The intuition behind this requirement is that if the threads do not use each other's intended DH exponentials then there could, in general, be related key attacks; the random oracle obviates this possibility.

CTXG $\mathsf{DHSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset$
$\mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)$

The earlier paper [36] overlooked the subtle difference between the $\mathsf{DHStrongSecretive}$ and $\mathsf{DHSecretive}$ predicates. Specifically, in order to prove the axiom **DH** sound without the random oracle model, it is necessary to ensure that both parties use only each other's DH exponentials to generate keys—a condition guaranteed by $\mathsf{DHStrongSecretive}$, but not $\mathsf{DHSecretive}$ or the variant considered in [36]. The notion of (x, y)-DHSafe described in Chapter 2 corresponds to the predicate $\mathsf{DHStrongSecretive}$.

To provide some sense of the soundness proofs, we sketch the proof for the **CTXGS** axiom. The axiom is sound if the set (given by the semantics)

$$\begin{aligned}&[\![\mathsf{DHStrongSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k)\\ &\supset \mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)]\!](T, D, \epsilon)\end{aligned}$$

includes almost all traces in the set T generated by any probabilistic poly-time adversary $\mathcal{A}$. Assume that this is not the case: Let E be the event that an honest principal decrypts a ciphertext c with the key k such that c was not produced by X or Y by encryption with the key k; there exists an adversary $\mathcal{A}$ who forces E to occur in a non-negligible number of traces. Using $\mathcal{A}$, we will construct an adversary $\mathcal{A}'$ who breaks DDH, thereby arriving at a contradiction.

Suppose $\mathcal{A}'$ is given a DDH instance (g^a, g^b, g^c). It has to determine whether $c = ab$. Let the DH nonces used by X, Y be x, y respectively. $\mathcal{A}'$ simulates execution of the protocol to $\mathcal{A}$ by using g^a, g^b as the computational representations of g^x, g^y respectively. Whenever a symbolic step ($k' :=$ `dhkeygen` $m, x;$) comes up, $\mathcal{A}'$ behaves in the following manner: since $\mathsf{DHStrongSecretive}(X, Y, k)$ holds, m has to be equal to g^b, then k' is assigned the value g^c; Likewise for the action ($k' :=$ `dhkeygen` $m, y;$). After the protocol simulation, if the event E has occurred then output "$c = ab$", otherwise output "$c \neq ab$". The advantage of $\mathcal{A}'$ in winning the DDH game is:

$$\mathbf{Adv}^{\mathrm{DDH}}(\mathcal{A}') = \Pr[E | c = ab] - \Pr[E | c \neq ab]$$

By the assumption about $\mathcal{A}$, the first probability is non-negligible. The second probability is negligible because the encryption scheme is INT-CTXT secure. Hence the advantage of $\mathcal{A}'$ in breaking DDH is non-negligible. The SendDHGood predicate that DHStrongSecretive implies, ensures that the protocol simulation can be carried out consistently. Intuitively, this is ensured as long as the protocol simulator has to manipulate received messages, g^x, g^y (but not x, y directly) and key messages with g^{xy} to construct messages to be sent out. Axioms $\mathbf{DH0} - \mathbf{6}$ are used to formally establish that the protocol has this property.

4.1.2 Secretive Protocols

In this section, we adapt the concept of *secretive protocol,* a trace-based condition implying computational secrecy (Chapter 2), to permit keys generated from DHKE. While the proof rules remain identical, the soundness proofs are significantly different and involve a reduction to a multi-scheme IND-CCA game that we introduce in Section 4.1.3. This definition allows the use of multiple encryption schemes keyed using randomly generated keys or keys output from a DHKE. A secretive protocol with respect to a nonce s and set of keys $\mathcal{K}$ is a protocol which generates *secretive traces*, defined below, with overwhelming probability.

Definition 26 (Secretive Trace) *A trace is a* secretive trace *with respect to s and $\mathcal{K}$ if the following properties hold for every thread belonging to honest principals:*

- *a thread which generates nonce s, ensures that it is encrypted with a key k in the set $\mathcal{K}$ in any message sent out.*
- *whenever a thread decrypts a message with a key k in $\mathcal{K}$, which was produced by encryption with key k by an honest party, and parses the decryption, it ensures that the results are encrypted with some key k' with $k' \in \mathcal{K}$ in any message sent out.*

To account for DH keys in the set $\mathcal{K}$, we wish to establish that DH keys are used in a "safe" manner by the protocol, formally captured by the predicate DHStrongSecretive.

Following Chapter 2, the predicate $\mathsf{Good}(X, m, s, \mathcal{K})$ asserts that the thread X constructed the term m in accordance with the rules allowing a *secretive protocol* with respect to nonce s and set of keys $\mathcal{K}$ to send out m. The formula $\mathsf{SendGood}(X, s, \mathcal{K})$ asserts that all messages that thread X sends out are good and $\mathsf{Secretive}(s, \mathcal{K})$ asserts that all honest threads only send out good messages. The axioms characterizing these predicates are same as in [66] and are omitted here. The induction rule $\mathbf{IND}_{GOOD}$ states that if all honest threads executing some basic sequence (i.e. a fragment of a role pausing before the next receive, denoted P) in the protocol (denoted $\mathcal{Q}$) locally construct good messages to be sent out, given that they earlier also did so, then we can conclude $\mathsf{Secretive}(s, \mathcal{K})$. A set of basic sequences (BS) of a role is any partition of the sequence of actions in a role such that if any element sequence has a `receive` then its only at its begining.

$$\mathbf{IND}_{GOOD} \quad \begin{array}{l} \forall \rho \in \mathcal{Q}. \forall P \in BS(\rho). \\ \dfrac{\mathsf{SendGood}(X, s, \mathcal{K})\ [P]_X\ \Phi \supset \mathsf{SendGood}(X, s, \mathcal{K})}{\mathcal{Q} \vdash \Phi \supset \mathsf{Secretive}(s, \mathcal{K})} \ (*) \\ (*)\text{: } [P]_X \text{ does not capture free variables in } \Phi,\ \mathcal{K},\ s, \\ \text{and } \Phi \text{ is a prefix closed trace formula.} \end{array}$$

Now we relate the concept of a secretive protocol, which is trace-based, to complexity theoretic notions of security. We define a level-0 key to be either a pre-shared secret, a public key or a DH Key. To apply the results here the DHStrongSecretive property has to hold for a DH key k for some pair of honest threads. A nonce is established to be a level-1 key when the protocol is proved to be a *secretive protocol* with respect to the nonce and a set of level-0 keys. This concept is extended further to define level-2 keys and so on.

We recall some predicates from Chapter 3 which are adapted here. For a set of keys $\mathcal{K}$ of levels ≤ 1, $\mathcal{B}(\mathcal{K})$ is the union of all the level-0 keys in $\mathcal{K}$ and the union of all the level-0 keys protecting the level-1 keys in $\mathcal{K}$. The formula $\mathsf{InInitSet}(X, s, \mathcal{K})$ asserts X is either the generator of nonce s or a possessor of some key in $\mathcal{B}(\mathcal{K})$.

$\mathsf{GoodInit}(s, \mathcal{K})$ asserts that all such threads belong to honest principals. Formally,

$$\mathsf{InInitSet}(X, s, \mathcal{K}) \equiv \exists k \in \mathcal{B}(\mathcal{K}).\, \mathsf{Possess}(X, k) \vee \mathsf{New}(X, s)$$
$$\mathsf{GoodInit}(s, \mathcal{K}) \equiv \forall X.\, (\mathsf{InInitSet}(X, s, \mathcal{K}) \supset \mathsf{Honest}(\hat{X}))$$

The formula $\mathsf{GoodKeyFor}$ lets us state that secrets established by secretive protocols, where possibly the secrets are also used as keys, are good keys against everybody except the set of principals who either generated the secret or are in possession of a key protecting the secret. For level-0 keys which we want to claim as being possessed only by honest principals we use the formula $\mathsf{GoodKey}$.

$$\mathsf{GoodKeyFor}(s, \mathcal{K}) \equiv \forall X.\, (\mathsf{GoodKeyAgainst}(X, s) \vee \mathsf{InInitSet}(X, s, \mathcal{K}))$$
$$\mathsf{GoodKey}(k) \equiv \forall X.\, (\mathsf{Possess}(X, k) \supset \mathsf{Honest}(\hat{X}))$$

For protocols employing an IND-CCA secure encryption scheme, the following axiom is sound:

$$\mathbf{GK} \quad \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \Rightarrow \mathsf{GoodKeyFor}(s, \mathcal{K})$$

If the encryption scheme is both IND-CCA and INT-CTXT secure then following axioms are sound:

$$\mathbf{CTX0} \quad \mathsf{GoodKey}(k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset \exists X.\, \mathsf{SymEnc}(X, m, k), \text{ for level-0 key } k.$$

$$\mathbf{CTXL} \quad \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \wedge \mathsf{SymDec}(Z, E_{sym}[s](m), s) \supset \exists X.\, \mathsf{SymEnc}(X, m, s)$$

The following axiom states that if a protocol is secretive with respect to s and $\mathcal{K}$, then the only keys, under which a message containing s openly is found encrypted in

$\mathsf{SendDHGood}(X, x)$	$\equiv \forall m.\ \mathsf{Send}(X, m) \supset \mathsf{DHGood}(X, m, x)$
$\mathsf{DHSecretive}(X, Y, k)$	$\equiv \exists x, y.\ \mathsf{New}(X, x) \wedge \mathsf{SendDHGood}(X, x) \wedge$ $\mathsf{New}(Y, y) \wedge \mathsf{SendDHGood}(Y, y) \wedge \mathsf{KeyGen}(X, k, x, g^y)$
$\mathsf{DHStrongSecretive}(X, Y, k)$	$\equiv \exists x, y.\ \mathsf{New}(X, x) \wedge \mathsf{SendDHGood}(X, x) \wedge$ $\mathsf{New}(Y, y) \wedge \mathsf{SendDHGood}(Y, y) \wedge \mathsf{KeyGen}(X, k, x, g^y) \wedge$ $(\mathsf{Exp}(X, gy, x) \supset gy = g^y) \wedge (\mathsf{Exp}(Y, gx, y) \supset gx = g^x)$
$\mathsf{SharedSecret}(X, Y, k)$	$\equiv \forall Z.\ \mathsf{GoodKeyAgainst}(Z, k) \vee Z = X \vee Z = Y$

Table 4.1: Summary of New Definitions

a "good" message, are in the set $\mathcal{K}$:

$$\textbf{SDEC} \quad \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{SymDec}(X, E_{sym}[k](m), k) \wedge$$
$$\mathsf{Good}(X, E_{sym}[k](m), s, \mathcal{K}) \wedge \mathsf{ContainsOpen}(m, s) \supset k \in \mathcal{K}$$

The new definitions in this chapter are summarized in Table 4.1 and the new axioms and proof rules are summarized in Table 4.2.

The **soundness theorem** is proved by showing that every axiom is a valid formula and that all proof rules preserve validity. The soundness proofs for the four axioms above are sketched in following sections; they proceed by reduction to the multiple encryption scheme game defined in the next section.

Theorem 17 (Soundness) $\forall \mathcal{Q}, \varphi.$ *if* $\mathcal{Q} \vdash \varphi$ *then* $\mathcal{Q} \models \varphi$

4.1.3 Joint Security of Multiple Encryption Schemes

A public-key encryption scheme $\mathcal{ES}$ is a triplet $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ such that $\mathcal{KG}(I)$ generates a pair of keys (ek, dk), where I is some initial information, ek is the public key and dk is the private key, and $\mathcal{E}$ and $\mathcal{D}$ are the encryption and decryption functions respectively. In [13], Bellare, Boldyreva and Micali analyzed the security of a single public-key encryption scheme in a setting where more than one independent keys are

DH0	$\mathsf{DHGood}(X,a,x)$, for a of any atomic type, except nonce, *viz.* name or key
DH1	$\mathsf{New}(Y,n) \wedge n \neq x \supset \mathsf{DHGood}(X,n,x)$
DH2	$[\texttt{receive}\ m;]_X\ \mathsf{DHGood}(X,m,x)$
DH3	$[m := \texttt{expg}\ x;]_X\ \mathsf{DHGood}(X,m,x)$
DH4	$\mathsf{DHGood}(X,m_0,x) \wedge \mathsf{DHGood}(X,m_1,x)\ [m := m_0.m_1;]_X\ \mathsf{DHGood}(X,m,x)$
DH5	$\mathsf{DHGood}(X,m,x)\ [m' := \texttt{symenc}\ m,k;]_X\ \mathsf{DHGood}(X,m',x)$
DH6	$\mathsf{DHGood}(X,m,x)\ [m' := \texttt{hash}\ m;]_X\ \mathsf{DHGood}(X,m',x)$
SDH0	$\mathsf{Start}(X) \supset \mathsf{SendDHGood}(X,x)$
SDH1	$\mathsf{SendDHGood}(X,x)\ [\mathsf{a}]_X\ \mathsf{SendDHGood}(X,x)$, where a is not a send action
SDH2	$\mathsf{SendDHGood}(X,x)\ [\texttt{send}\ m;]_X\ \mathsf{DHGood}(X,m,x) \supset \mathsf{SendDHGood}(X,x)$
DH	$\mathsf{DHStrongSecretive}(X,Y,k) \Rightarrow \mathsf{SharedKey}(X,Y,k)$
CTXGS	$\mathsf{DHStrongSecretive}(X,Y,k) \wedge \mathsf{SymDec}(Z,E_{sym}[k](m),k) \supset$ $\mathsf{SymEnc}(X,m,k) \vee \mathsf{SymEnc}(Y,m,k)$
CTXG	$\mathsf{DHSecretive}(X,Y,k) \wedge \mathsf{SymDec}(Z,E_{sym}[k](m),k) \supset$ $\mathsf{SymEnc}(X,m,k) \vee \mathsf{SymEnc}(Y,m,k)$
SDEC	$\mathsf{Secretive}(s,\mathcal{K}) \wedge \mathsf{SymDec}(X,E_{sym}[k](m),k) \wedge$ $\mathsf{Good}(X,E_{sym}[k](m),s,\mathcal{K}) \wedge \mathsf{ContainsOpen}(m,s) \supset k \in \mathcal{K}$

Table 4.2: Summary of New Axioms and Proof Rules

used. The security of an encryption scheme is defined in terms of a game between an adversary and a challenger. In the *chosen plaintext* (IND-CPA) setting, the adversary has access to a *left-or-right* encryption oracle $\mathcal{E}_{ek}(LR(\cdot,\cdot,b))$, which takes a pair of equal length messages m_0, m_1 from the adversary and returns the encryption of m_b with the key ek, the bit b being unknown to the adversary. In the *chosen ciphertext* (IND-CCA) setting, the adversary has, in addition, access to a decryption oracle $\mathcal{D}_{dk}(\cdot)$, with the caveat that it cannot query for the decryption of a ciphertext it received as an answer to a previous encryption oracle query.

In this section, we extend their definition to settings involving multiple encryption schemes. Consider a sequence of n, not necessarily distinct, encryption schemes $\langle \mathcal{ES}^i \mid 1 \leq i \leq n \rangle$, possibly consisting of public-key and symmetric-key encryption schemes with either pre-shared keys or setup by a Diffie-Hellman exchange. For notational uniformity we define $ek_i = dk_i$ for symmetric key schemes, both equal to the secret key. For Diffie-Hellman schemes, $ek_i = dk_i = keygen(g^{xy})$ where g^x and g^y are the public DH values. Let DH be the set of Diffie-Hellman public values (g^x, g^y) for those keys which are generated by a DH exchange and PK be the set of public-keys among the ek_i's. In the *multi-scheme* setting we let the adversary have access to n encryption and decryption oracles with their corresponding public informations (PK and DH), all using the *same* challenger bit b for encryption. Security in this setting is defined below.

Definition 27 (Multi Scheme Indistinguishability) *The experiment MS-IND-CCA, for adversary A, is defined as:*

```
Experiment $\mathbf{Exp}^{MS\text{-}IND\text{-}CCA}_{\langle\mathcal{ES}\rangle,I}(A,b)$
    For $i = 1,\cdots,n$ do $(ek_i, dk_i) \leftarrow \mathcal{KG}^i(I)$ EndFor
    $d \leftarrow A^{\mathcal{E}^1_{ek_1}(LR(\cdot,\cdot,b)),\ldots,\mathcal{E}^n_{ek_n}(LR(\cdot,\cdot,b)),\mathcal{D}^1_{dk_1}(\cdot),\ldots,\mathcal{D}^n_{dk_n}(\cdot)}(I, PK, DH)$
    Return $d$
```

A query to any LR oracle consists of two messages of equal *length and that for each $i = 1, \ldots, n$ adversary A does not query $\mathcal{D}_{dk_i}(\cdot)$ on an output of $\mathcal{E}^i_{ek_i}(LR(\cdot,\cdot,b))$. The* advantage *of A is defined as:*

$$\mathbf{Adv}^{\textit{MS-IND-CCA}}_{\langle\mathcal{ES}\rangle,I}(A) = \Pr[\mathbf{Exp}^{\textit{MS-IND-CCA}}_{\langle\mathcal{ES}\rangle,I}(A,0)=0] - \Pr[\mathbf{Exp}^{\textit{MS-IND-CCA}}_{\langle\mathcal{ES}\rangle,I}(A,1)=0]$$

The sequence of encryption schemes $\langle\mathcal{ES}^i \mid 1 \leq i \leq n\rangle$ is MS-IND-CCA *secure if the advantage of any probabilistic poly-time adversary A is negligible in the security parameter.*

The definition of MS-IND-CPA is similar, with the decryption oracles dropped. We prove that individual security of the encryption schemes implies joint security.

Theorem 18 (IND-CPA(CCA) → MS-IND-CPA(CCA)) *If encryption schemes $\mathcal{ES}_1, \mathcal{ES}_2, \ldots, \mathcal{ES}_n$ are individually IND-CPA(CCA)secure, then the sequence of schemes $\langle\mathcal{ES}_1, \mathcal{ES}_2, \ldots, \mathcal{ES}_n\rangle$ is MS-IND-CPA(CCA) secure.*

We will proceed in a few stages to arrive at the final proofs of Theorem 17 (**Soundness**) and Theorem 18 (**IND-CPA(CCA) → MS-IND-CPA(CCA)**). We begin with the second theorem since it is essential to the proof of the first. First, we define a *DH-CCA* game which defines the security of encryption using keys established by DHKE and reduce this to the standard security notions of Decisional Diffie-Hellman (DDH) and Indistinguishability under Chosen Ciphertext Attack (IND-CCA). Finally, we proceed to reduce the joint multi-scheme security (MS-IND-CCA) defined in this chapter to a combination of IND-CCA for symmetric and public-key encryption schemes and DH-CCA.

Definition 28 (DH encryption advantage) *Let $\mathcal{G}$ be a group and $g \in \mathcal{G}$. A DH-CPA challenger chooses $g^x, g^y \xleftarrow{\$} \mathcal{G}, b \xleftarrow{\$} \{0,1\}$ and provides g^x, g^y and oracles $\mathcal{E}_k(\cdot,\cdot,b)$, where $k = keygen(g^{xy})$, to an adversary algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit b'. The DH-CPA advantage of algorithm $\mathcal{A}$ is:*

$$\mathbf{Adv}^{\textit{DH-CPA}}(\mathcal{A}) = \Pr[b'=0|b=0] - \Pr[b'=0|b=1]$$

$\mathbf{Adv}^{\textit{DH-CCA}}(\mathcal{A})$ *is defined analogously with the addition of a decryption oracle $\mathcal{D}_k(\cdot)$ which only decrypts messages not produced by the encryption oracle.*

Theorem 19 (DH encryption security) *Let $\langle \mathcal{E}, \mathcal{D}, \mathcal{K} \rangle$ be an IND-CPA (CCA) secure symmetric key encryption scheme. Let $\mathcal{G}$ be a group in which DDH is hard. Assume that keygen(h), $h \xleftarrow{\$} \mathcal{G}$ has the same probability distribution as the key space. A PPT adversary having $g^x, g^y \xleftarrow{\$} \mathcal{G}$ has negligible advantage in winning an IND-CPA (CCA) game with a challenger using the key k.*

Proof. We prove it for the case of IND-CCA, CPA is analogous. Assume on the contrary that such a PPT adversary $\mathcal{A}$ exists. We will construct a DDH adversary $\mathcal{A}'$ using $\mathcal{A}$. $\mathcal{A}'$ is given (g^x, g^y, g^z) and has to determine whether $z = xy$. $\mathcal{A}'$ then proceeds as follows:

```
$\mathcal{A}' : b \xleftarrow{\$} \{0,1\}$
$\mathcal{A}' \longrightarrow \mathcal{A} : g^x, g^y$
do
    $\mathcal{A} \longrightarrow \mathcal{A}'$ : encrypt $m_0, m_1 \mapsto \mathcal{A}' \longrightarrow \mathcal{A} : enc(m_b, g^z)$
    $\mathcal{A} \longrightarrow \mathcal{A}'$ : decrypt $c \mapsto \mathcal{A}' \longrightarrow \mathcal{A} : dec(c, g^z)$, $c$ was not an encryption query
until
    $\mathcal{A} \longrightarrow \mathcal{A}'$ : output $b'$
if $b' = b$
then
    $\mathcal{A}'$ : output '$z = xy$'
else
    $\mathcal{A}'$ : output '$z \neq xy$'
```

Now,

$$\begin{aligned}\mathbf{Adv}^{\text{DDH}}(\mathcal{A}') &= \Pr[b' = b | z = xy] - \Pr[b' = b | z \neq xy] \\ &= \frac{1}{2}\left[\mathbf{Adv}^{\text{DH-CCA}}(\mathcal{A}) - \mathbf{Adv}^{\text{IND-CCA}}(\mathcal{A})\right]\end{aligned}$$

Therefore,

$$\mathbf{Adv}^{\text{DH-CCA}}(\mathcal{A}) = 2\mathbf{Adv}^{\text{DDH}}(\mathcal{A}') + \mathbf{Adv}^{\text{IND-CCA}}(\mathcal{A})$$

Since both the advantages on the RHS are negligible, so is the LHS. □

Definition 29 (Single Scheme Indistinguishability) *The experiments IND-CPA and IND-CCA are defined as:*

Experiment $\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{ES},I}(A_{cpa}, b)$	**Experiment** $\mathbf{Exp}^{\text{IND-CCA}}_{\mathcal{ES},I}(A_{cca}, b)$
$(ek, dk) \leftarrow \mathcal{K}(I)$	$(ek, dk) \leftarrow \mathcal{K}(I)$
$d \leftarrow A_{cpa}^{\mathcal{E}_{ek}(LR(\cdot,\cdot,b))}(I, PK, DH)$	$d \leftarrow A_{cca}^{\mathcal{E}_{ek}(LR(\cdot,\cdot,b)),\mathcal{D}_{dk}(\cdot)}(I, PK, DH)$
`Return` d	`Return` d

A query to the LR oracle consists of two messages of equal *length and that adversary A_{cca} does not query $\mathcal{D}_{dk}(\cdot)$ on an output of $\mathcal{E}_{ek}(LR(\cdot,\cdot,b))$. The* advantage *of A_{cpa}, and the* advantage *of A_{cca}, respectively, as follows:*

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{ES},I}(A_{cpa}) = \Pr[\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{ES},I}(A_{cpa}, 0) = 0] - \Pr[\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{ES},I}(A_{cpa}, 1) = 0]$$
$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{ES},I}(A_{cca}) = \Pr[\mathbf{Exp}^{\text{IND-CCA}}_{\mathcal{ES},I}(A_{cca}, 0) = 0] - \Pr[\mathbf{Exp}^{\text{IND-CCA}}_{\mathcal{ES},I}(A_{cca}, 1) = 0]$$

Theorem 20 (Multi Scheme Security) *Let $\langle\mathcal{ES}\rangle = \langle\mathcal{ES}^1, \ldots, \mathcal{ES}^n\rangle$ be a sequence of n encryption schemes. Let I be some initial information string. Then, for any adversary A,*

$$\mathbf{Adv}^{\text{MS-IND-CPA}}_{\langle\mathcal{ES}\rangle,I}(A) \leq \sum_{i=1}^{n} \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{ES}^i,I}(A)$$
$$\mathbf{Adv}^{\text{MS-IND-CCA}}_{\langle\mathcal{ES}\rangle,I}(A) \leq \sum_{i=1}^{n} \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{ES}^i,I}(A)$$

Proof. We start by defining the following n hybrid experiments:

```
Experiment ExpH_l   $[0 \leq l \leq n]$
  For $i = 1, \ldots, n$ do $(ek_i, dk_i) \leftarrow \mathcal{K}^i(I)$ EndFor
  Run $A$ replying to oracle queries as follows:
    $A \rightarrow (i, m_0, m_1) \quad [1 \leq i \leq n]$
    If $i \leq l$ then $C \leftarrow \mathcal{E}^i_{ek_i}(m_0)$ EndIf
    If $i > l$ then $C \leftarrow \mathcal{E}^i_{ek_i}(m_1)$ EndIf
    $A \leftarrow C$
  Eventually $A$ halts outputting a bit $d$
  Return $d$
```

We now build individual adversaries B for each encryption scheme in such a way that advantage of each one is the difference of consecutive hybrid experiments as described above:

```
Adversary $B_{A,j}^{\mathcal{E}^j_{ek}(LR(\cdot,\cdot,b))}$
  For $i \in \{1, \ldots, j-1, j+1, \ldots, n\}$
   do $(ek_i, dk_i) \leftarrow \mathcal{K}^i(I)$ EndFor
  Run $A$ replying to oracle queries as follows:
    $A \rightarrow (i, m_0, m_1) \quad [1 \leq i \leq n]$
    If $i < j$ then $C \leftarrow \mathcal{E}^i_{ek_i}(m_0)$ EndIf
    If $i > j$ then $C \leftarrow \mathcal{E}^i_{ek_i}(m_1)$ EndIf
    If $i = j$ then $C \leftarrow \mathcal{E}^j_{ek}(LR(m_0, m_1, b))$ EndIf
    $A \leftarrow C$
  Eventually $A$ halts outputting a bit $d$
  Return $d$
```

Therefore,

$$\Pr[B_{A,j}^{\mathcal{E}^j_{ek}(LR(\cdot,\cdot,b))} = 0|b = 1] = \Pr[\mathbf{ExpH}_{j-1} = 0]$$
$$\Pr[B_{A,j}^{\mathcal{E}^j_{ek}(LR(\cdot,\cdot,b))} = 0|b = 0] = \Pr[\mathbf{ExpH}_{j} = 0]$$

Hence,

$$\begin{aligned}\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{ES}^j,I}(B_{A,j}) &= \Pr[B_{A,j}^{\mathcal{E}^j_{ek}(LR(\cdot,\cdot,b))} = 0|b = 0] - \\ &\quad \Pr[B_{A,j}^{\mathcal{E}^j_{ek}(LR(\cdot,\cdot,b))} = 0|b = 1] \\ &= \Pr[\mathbf{ExpH}_j = 0] - \Pr[\mathbf{ExpH}_{j-1} = 0]\end{aligned}$$

So we have:

$$\begin{aligned}\sum_{j=1}^{n} \mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{ES}^j,I}(B_{A,j}) &= \sum_{j=1}^{n} \left[\Pr[\mathbf{ExpH}_j = 0] - \Pr[\mathbf{ExpH}_{j-1} = 0]\right] \\ &= \Pr[\mathbf{ExpH}_n = 0] - \Pr[\mathbf{ExpH}_0 = 0]\end{aligned}$$

Observe that this difference is the advantage $\mathbf{Adv}^{\text{MS-IND-CPA}}_{\langle\mathcal{ES}\rangle,I}(A)$. So we have constructed adversaries against the individual encryption schemes such that A's advantage is the sum of their advantages. So maximized over all adversaries A, the best advantage is at most equal to the sum of the best individual advantages.

The proof is analogous for MS-IND-CCA - there we additionally use decryption oracles. Hence the theorem. □

Proof of Theorem 18 (**IND-CPA(CCA) → MS-IND-CPA(CCA)**). Follows from theorem 20. □

Proof of Theorem 17 (**Soundness**). We give soundness proofs of the new axioms in this chapter.

Axioms DH0-DH6, SDH0-SDH2: Straightforward from the semantics of DHGood and the definition of the formula $\mathsf{SendDHGood}$.

Axiom DH:

$$\mathsf{DHStrongSecretive}(X, Y, k) \Rightarrow \mathsf{SharedKey}(X, Y, k)$$

The axiom is sound if the semantics of the formula $[\![\mathsf{DHStrongSecretive}(X, Y, k) \Rightarrow \mathsf{SharedKey}(X, Y, k)]\!](T, D, \epsilon)$ is an overwhelming fraction of the set of traces T generated by any probabilistic poly-time adversary $\mathcal{A}$. This means, for all sufficient large security parameters η, if a trace satisfies the predicate
$\mathsf{DHStrongSecretive}(X, Y, k)$, then no distinguisher can win an IND-CCA challenge with key k played at the end of interaction with the protocol with non-negligible advantage.

Suppose $\mathcal{A}'$ is given a DDH instance (g^a, g^b, g^c). It has to determine whether $c = ab$. Given that $\mathsf{DHStrongSecretive}(X, Y, k)$ holds, let the DH nonces used by X, Y be x, y respectively. $\mathcal{A}'$ simulates execution of the protocol to $\mathcal{A}$ by using g^a, g^b as the computational representations of g^x, g^y respectively. Note that because of the DHStrongSecretive condition, only g^a and g^b are used in the construction of terms to be sent - the value of a and b themselves is not required. Whenever a symbolic step $(k' := \texttt{dhkeygen}\ m, x;)$ comes up, since $\mathsf{DHStrongSecretive}(X, Y, k)$ holds, m has to be equal to g^b. $\mathcal{A}'$ then assigns k' the value g^c; Likewise for the action $(k' := \texttt{dhkeygen}\ m, y;)$. Due to the DHStrongSecretive condition, g^c is only used as an encryption key. After the protocol simulation, $\mathcal{A}'$ chooses a bit $\hat{b}$ uniformly randomly and provides an IND-CCA challenger using the key g^c to $\mathcal{A}$. At the end, $\mathcal{A}$ outputs a guess b'. If $\hat{b} = b'$, $\mathcal{A}'$ outputs "$c = ab$", otherwise output "$c \neq ab$"

The advantage of $\mathcal{A}'$ in winning the DDH game is:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{DDH}}(\mathcal{A}') &= \Pr[\hat{b} = b' | c = ab] - \Pr[\hat{b} = b' | c \neq ab] \\ &= (\Pr[\hat{b} = b' | c = ab] - \frac{1}{2}) - (\Pr[\hat{b} = b' | c \neq ab] - \frac{1}{2})\end{aligned}$$

By the assumption about $\mathcal{A}$, the first quantity is non-negligible. The second quantity is negligible because the encryption scheme is IND-CCA secure. Hence the advantage of $\mathcal{A}'$ in breaking DDH is non-negligible.

Axiom CTXGS:

$$\mathsf{DHStrongSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset \\ \mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)$$

The axiom is sound if the semantics of the formula $[\![\mathsf{DHStrongSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset \mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)]\!](T, D, \epsilon)$ is an overwhelming fraction of the set of traces T generated by any probabilistic poly-time adversary $\mathcal{A}$. This means, for all sufficient large security parameters η, if a trace satisfies the predicate $\mathsf{DHStrongSecretive}(X, Y, k)$ and an honest principle decrypts a ciphertext $E_{sym}[k](m)$ with the key k, then with overwhelming probability $E_{sym}[k](m)$ was produced by X or Y by encryption with the key k.

Let E be the event that an honest principle decrypts a ciphertext c with the key k, such that c was not produced by X or Y by encryption with the key k. Consider, on the contrary, an adversary $\mathcal{A}$, such that in a non-negligible number of traces the event E takes place. Using $\mathcal{A}$, we will construct an adversary $\mathcal{A}'$ to the Decisional Diffie Hellman problem.

$\mathcal{A}'$ simulates the protocol to $\mathcal{A}$ exactly as in the proof of axiom **DH** above. After the protocol simulation, if the event E has occurred then output "$c = ab$", otherwise output "$c \neq ab$".

The advantage of $\mathcal{A}'$ in winning the DDH game is:

$$\mathbf{Adv}^{\mathrm{DDH}}(\mathcal{A}') = \Pr[E|c = ab] - \Pr[E|c \neq ab]$$

By the assumption about $\mathcal{A}$, the first probability is non-negligible. The second probability is negligible because the encryption scheme is INT-CTXT secure. Hence the advantage of $\mathcal{A}'$ in breaking DDH is non-negligible.

Axiom CTXG:

$$\mathsf{DHSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset \\ \mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)$$

The axiom is sound if the semantics of the formula

$$[\![\mathsf{DHSecretive}(X, Y, k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k)$$
$$\supset \mathsf{SymEnc}(X, m, k) \vee \mathsf{SymEnc}(Y, m, k)]\!](T, D, \epsilon)$$

is an overwhelming fraction of the set of traces T generated by any probabilistic poly-time adversary $\mathcal{A}$. This means, for all sufficient large security parameters η, if a trace satisfies the predicate $\mathsf{DHSecretive}(X, Y, k)$ and an honest principle decrypts a ciphertext $E_{sym}[k](m)$ with the key k, then with overwhelming probability $E_{sym}[k](m)$ was produced by X or Y by encryption with the key k.

Let E be the event that an honest principle decrypts a ciphertext c with the key k, such that c was not produced by X or Y by encryption with the key k. Consider, contrary to the semantics of the axiom, an adversary $\mathcal{A}$, such that in a non-negligible number of traces the event E takes place. Using $\mathcal{A}$, we will construct an adversary $\mathcal{A}'$ to the INT-CTXT security of the symmetric encryption scheme, assuming the Computational Diffie-Hellman (CDH) assumption holds and the key generation function is a Random Oracle $\mathcal{H}(\cdot)$.

Given that $\mathsf{DHSecretive}(X, Y, k)$ holds, let the DH nonces used by X, Y be x, y respectively. $\mathcal{A}'$ simulates execution of the protocol to $\mathcal{A}$ by using randomly generated a, b as the computational representations of x, y respectively. Whenever a symbolic step ($k' :=$ `dhkeygen` m, x;) comes up, $\mathcal{A}'$ behaves in the following manner:

- If the value of m is equal to g^b, then encryption by k' is performed by using the INT-CTXT challenger. This represents the use of the INT-CTXT challenger's key as the random oracle's answer $\mathcal{H}(g^{ab})$. Since the INT-CTXT challenger chooses a randomly generated key, the distribution over all possible keys would correspond to the distribution of a Random Oracle's reply to the value g^{ab}.

- As compared to the DHStrongSecretive condition, it is not necessary that the value of m is g^b if only the DHSecretive condition holds. For each other distinct value of m, $\mathcal{A}'$ generates a random r as the random oracle's answer $\mathcal{H}((\text{value of } m)^a)$ and uses this as the key.

Likewise for the symbolic step ($k' :=$ `dhkeygen` $m, y;$). Algorithm $\mathcal{A}'$ should ensure that the oracle queries of $\mathcal{A}$ are answered consistently with the above computations.

Since $\mathsf{DHSecretive}(X, Y, k)$ holds, the key generated from g^{ab} is only used as a key by X and Y. Therefore the encryption oracle is enough to simulate protocol actions. Also, since only g^a and g^b are used in the construction of terms to be sent, due to the computational Diffie-Hellman assumption (CDH), it is infeasible that $\mathcal{A}$ would query the random oracle with the actual value of g^{ab}, hence, $\mathcal{A}'$ does not have to know the value of the INT-CTXT challenger's key.

Thus $\mathcal{A}'$'s execution trace distribution is statistically indistinguishable from the protocol's execution with a random oracle for key generation. Therefore, $\mathcal{A}$ should cause the event E to occur non-negligibly often in this set of traces. When the event E occurs, it means $\mathcal{A}'$ comes up with a ciphertext not produced by the encryption oracle and hence wins against the INT-CTXT challenger.

Axioms GK, CTX0, CTXL:

$$\begin{array}{rl}
\textbf{GK} & \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \Rightarrow \mathsf{GoodKeyFor}(s, \mathcal{K}) \\
\textbf{CTX0} & \mathsf{GoodKey}(k) \wedge \mathsf{SymDec}(Z, E_{sym}[k](m), k) \supset \\
 & \quad \exists X.\ \mathsf{SymEnc}(X, m, k), \text{ for level-0 key } k. \\
\textbf{CTXL} & \mathsf{Secretive}(s, \mathcal{K}) \wedge \mathsf{GoodInit}(s, \mathcal{K}) \wedge \mathsf{SymDec}(Z, E_{sym}[s](m), s) \supset \\
 & \quad \exists X.\ \mathsf{SymEnc}(X, m, s)
\end{array}$$

The proofs proceed identically as in Chapter 3 with one important difference: the reductions are to the *multi-scheme IND-CCA* (MS-IND-CCA) game, formulated in this chapter, instead of multi-party IND-CCA. As a consequence, the axioms are applicable to secrets which are protected jointly by one or more of the following encryption schemes: public-key, symmetric-key with pre-shared keys and symmetric-key with keys exchanged by DHKE. One case of **GK** is sketched here for completeness: where $\mathcal{K}$ consists of only level-0 keys, i.e., keys not used as payloads. For the MS-IND-CCA definition to be applicable to a key k established by DHKE, it is required that the predicate $\mathsf{DHStrongSecretive}(X, Y, k)$ holds for k and honest threads X and

Y. The full soundness of **GK** also involves keys protected by encryption with other keys.

Assume that a probabilistic poly-time adversary $\mathcal{A}$ interacts with a secretive protocol with respect to nonce s and a set of level-0 keys $\mathcal{K}$. We will show that if $\mathcal{A}$ has non-negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key s, after the interaction then we can construct either a MS-IND-CCA adversary $\mathcal{A}_1$ or an IND-CCA adversary $\mathcal{A}_2$ with non-negligible advantages against the encryption scheme.

Adversary $\mathcal{A}_1$ has access to multi-party Left-or-Right encryption oracles $\mathcal{E}_{k_i}(LoR(\cdot,\cdot,b))$ parameterized by a bit b and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}$ (Following [13], $LoR(m_0, m_1, b)$ is a function which returns m_b). $\mathcal{A}_1$ will provide a simulation of the *secretive protocol* to $\mathcal{A}$ by using these oracles. $\mathcal{A}_1$ randomly chooses two nonces x_0, x_1 at the outset as alternate bit-string representations of s. Suppose $u(s, \cdots)$ is a term explicitly constructed from s. As $\mathcal{A}_1$ is simulating a *secretive protocol*, this term is to be encrypted with a key k in $\mathcal{K}$ to construct a message to be sent out to $\mathcal{A}$. In this case $\mathcal{A}_1$ asks the encryption oracle $(u(x_0, \cdots), u(x_1, \cdots))$ to be encrypted by k. If a message construction involves decryption with a key in $\mathcal{K}$, $\mathcal{A}_1$ first checks whether the term to be decrypted was produced by an encryption oracle - if not then the decryption oracle is invoked; if yes then $\mathcal{A}_1$ uses the corresponding encryption query as the decryption. In the second case the encryption query might have been of the form $(v(x_0, \cdots), v(x_1, \cdots))$. Following the definition of *secretive protocol*, terms constructed from this decryption will be re-encrypted with a key in $\mathcal{K}$ before sending out. Thus we note here that all such replies will be consistent to $\mathcal{A}$ with respect to any choice of b. The situation becomes different when encryption or decryption of a term is required with s as the key. In this case $\mathcal{A}_1$ encrypts or decrypts with x_0.

In the second phase, $\mathcal{A}_1$ uniformly randomly chooses a bit b' and provides oracles $\mathcal{E}_{x_0}(LoR(\cdot,\cdot,b'))$ and $\mathcal{D}_{x_0}(\cdot)$ to $\mathcal{A}$ for an IND-CCA game. $\mathcal{A}$ finishes by outputting a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs $d = 0$ else outputs $d = 1$. The advantage of $\mathcal{A}_1$ against the MS-IND-CCA challenger is:

$$\mathbf{Adv}^{\text{MS-IND-CCA}}(\mathcal{A}_1) = Pr[d = 0 | b = 0] - Pr[d = 0 | b = 1] \tag{4.1}$$

Observe that if $b = 0$ then s was consistently represented by x_0 in messages sent to $\mathcal{A}$. Hence, the first probability is precisely the probability of $\mathcal{A}$ winning an IND-CCA challenge with s as the key after interacting with a *secretive protocol* w.r.t. s and $\mathcal{K}$. We will now bound the second probability. We start by constructing a second adversary $\mathcal{A}_2$ which has all the keys in $\mathcal{K}$, randomly generates a nonce x_1 and has access to an encryption oracle $\mathcal{E}_{x_0}(LoR(\cdot,\cdot,b_1))$ and a decryption oracle $\mathcal{D}_{x_0}(\cdot)$. It has a similar behaviour towards $\mathcal{A}$ as $\mathcal{A}_1$ had except that when constructing terms with s, it uses x_1 but when required to encrypt or decrypt using s, it queries $\mathcal{E}_{x_0}(LoR(\cdot,\cdot,b_1))$ or $\mathcal{D}_{x_0}(\cdot)$. In the second phase, $\mathcal{A}_1$ uses the oracles $\mathcal{E}_{x_0}(LoR(\cdot,\cdot,b_1))$ and $\mathcal{D}_{x_0}(\cdot)$ to provide the IND-CCA challenger to $\mathcal{A}$. $\mathcal{A}$ finishes by outputting a bit d_1. $\mathcal{A}_2$ outputs d_1. We observe here that if $b = 1$ for the earlier LoR oracle, it makes no difference to the algorithm $\mathcal{A}$ whether it is interacting with $\mathcal{A}_1$ or $\mathcal{A}_2$. Thus we have:

$$(1/2)\mathbf{Adv}^{\text{IND-CCA}}(\mathcal{A}_2) = \Pr[d_1 = b_1] - 1/2 = \Pr[d = 0|b = 1] - 1/2 \tag{4.2}$$

By the equations 4.1 and 4.2 we have:

$$\Pr[d = 0|b = 0] - \frac{1}{2} = \mathbf{Adv}^{\text{MS-IND-CCA}}(\mathcal{A}_1) + \frac{1}{2}\mathbf{Adv}^{\text{IND-CCA}}(\mathcal{A}_2)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done.

Axiom SDEC:

$$\begin{aligned}&\mathsf{Secretive}(s,\mathcal{K}) \wedge \mathsf{SymDec}(X, E_{sym}[k](m), k) \wedge \mathsf{Good}(X, E_{sym}[k](m), s, \mathcal{K}) \wedge \\ &\quad \mathsf{ContainsOpen}(m, s) \supset k \in \mathcal{K}\end{aligned}$$

The proof of soundness of this axiom depends on the use of a *discrete action bilateral simulator* described in Chapter 2. The intuition behind the proof is as follows: Assume, contrary to the implication, that $k \notin \mathcal{K}$. Then, since $E_{sym}[k](m)$ is a "good" term, its decryption with key k, i.e., m would also be a "good" term and hence should not contain s in the open. As is done in the proof of Lemma 2, this intuition is exploited in a reduction to an MS-IND-CCA game. □

4.2 Kerberos with DHINIT

In this section, we formally model Kerberos with DHINIT and prove that it satisfies computational authentication and secrecy properties under standard assumptions about the cryptographic primitives. Authentication proofs for each stage of Kerberos rely on the secrecy guarantees of keys set up in earlier stages, while the secrecy proofs similarly rely on previously proved authentication guarantees, an alternation first pointed out in [28]. Since the later stages of DHINIT are the same as those of Basic Kerberos [52], we obtain proofs for the complete protocol by appealing to security proofs and composition theorems in a compatible setting [66].

We find, perhaps surprisingly, that the KAS is *not* authenticated to the client after the first stage and suggest a fix to the protocol to avoid this problem. Our counterexample is similar in flavor to the attack found on Kerberos V5 with public-key initialization by [27]. In addition, we use an axiom that relies on *random oracles* to complete the proof of the security properties. We also develop an alternative proof, using only axioms that hold in the standard model, for a variant of the protocol that requires the KAS to sign both the Diffie-Hellman exponentials. We leave open whether this discrepancy arises from a security flaw in DHINIT or a limitation of our current proof.

4.2.1 Modeling the Protocol

The Kerberos protocol involves four roles—the Client, the Kerberos Authentication Server (KAS), the Ticket Granting Server (TGS), and the application server. The KAS and the TGS share a long term symmetric key as do the TGS and the application server. Mutual authentication and key establishment between the client and the application server is achieved by using this chain of trust. We write $k_{X,Y}^{type}$ to refer to long term symmetric keys, where X and Y are the principals sharing the key and *type* indicates their roles, e.g. $t \rightarrow k$ for TGS and KAS and $s \rightarrow t$ for application server and TGS. Kerberos runs in three stages with the client role participating in all three. The complete formal description of the protocol is given in Table 4.2.1.

The client C and KAS K carry out a Diffie-Hellman key exchange protocol authenticated by digital signatures to set up a key $AKey$ to be used as a session key between the client and the TGS in the next stage. (In Basic Kerberos, this phase is simpler; it relies on a preshared key between C and K.) The first few actions of the client are explained as follows: it generates three random numbers $n_1, \tilde{n_1}, x$ using `new` actions. It then generates the Diffie-Hellman exponential gx and sends a message to the KAS K containing its signature over the exponential and a few other fields including the identities of the TGS $\hat{T}$ and itself. In the second stage, the client gets a new session key ($SKey$ - Service Key) and a service ticket (st) to converse with the application server S which takes place in the third stage. The control flow of Kerberos exhibits a staged architecture where once one stage has been completed successfully, the subsequent stages can be performed multiple times or aborted and started over for handling errors.

4.2.2 Security Properties and Proofs

Table 4.4 lists the authentication and secrecy properties of Kerberos with DHINIT that we want to prove. The authentication properties are of the form that a message of a certain format was indeed sent by some thread of the expected principal. The secrecy properties state that a putative secret is a good key for certain principals. For example, $AUTH_{kas}^{client}$ states that when C finishes executing the **Client** role, some thread of $\hat{K}$ indeed sent the expected message with probability asymptotically close to one; SEC_{akey}^{client} states that the authorization key is "good" after execution of the **Client** role by C. The other security properties are analogous. More specifically, $\mathsf{GoodKeyAgainst}(\mathsf{X}, \mathsf{k})$ [36] intuitively means that if k were used instead of a random key to key an IND-CCA encryption scheme, then the advantage of X in the corresponding security game would be negligible. The motivation for using this definition is that stronger conditions such as key indistinguishability fail to hold as soon as the key is used; key indistinguishability is

```
**Client** $= (C, \hat{K}, \hat{T}, \hat{S}, t)$ [
  new $n_1$; new $\tilde{n_1}$;
  new $x$; $gx :=$ expg $x$;
  $chksum :=$ hash $\hat{C}.\hat{T}.n_1$;
  $sigc :=$ sign "$Auth$".$chksum.\tilde{n_1}.gx, sk_C$;
  send $Cert_C.sigc.\hat{C}.\hat{T}.n_1$;

  receive $Cert_K.sigk.\hat{C}.tgt.enc_{kc}$;
  verify $sigk$, "$DHKey$".$gy.\tilde{n_1}, vk_K$;
  $k :=$ dhkeygen $gy, x$;
  $text_{kc} :=$ symdec $enc_{kc}, k$;
  match $text_{kc}$ as $AKey.n_1.\hat{T}$;

  ··· stage boundary ···

  new $n_2$;
  $enc_{ct} :=$ symenc $\hat{C}, AKey$;
  send $tgt.enc_{ct}.\hat{C}.\hat{S}, n_2$;

  receive $\hat{C}.st.enc_{tc}$;
  $text_{tc} :=$ symdec $enc_{tc}, AKey$;
  match $text_{tc}$ as $SKey.n_2.\hat{S}$;

  ··· stage boundary ···

  $enc_{cs} :=$ symenc $\hat{C}.t, SKey$;
  send $st.enc_{cs}$;

  receive $enc_{sc}$;
  $text_{sc} :=$ symdec $enc_{sc}, SKey$;
  match $text_{sc}$ as $t$;
  $]_C$
```

```
**KAS** $= (K)$ [
  receive $Cert_C.sigc.\hat{C}.\hat{T}.n_1$;
  verify $sigc$, "$Auth$".$chksum.\tilde{n_1}.gx, vk_C$;
  $chk :=$ hash $\hat{C}.\hat{T}.n_1$;
  match $chk$ as $chksum$;
  new $AKey$;
  new $y$; $gy :=$ expg $y$;
  $k :=$ dhkeygen $gx, y$;
  $sigk :=$ sign "$DHKey$".$gy.\tilde{n_1}, sk_K$;
  $tgt :=$ symenc $AKey.\hat{C}, k_{T,K}^{t \to k}$;
  $enc_{kc} :=$ symenc $AKey.n_1.\hat{T}, k$;
  send $Cert_K.sigk.\hat{C}.tgt.enc_{kc}$;
  $]_K$

**TGS** $= (T, \hat{K})$ [
  receive $tgt.enc_{ct}.\hat{C}.\hat{S}.n_2$;
  $text_{tgt} :=$ symdec $tgt, k_{T,K}^{t \to k}$;
  match $text_{tgt}$ as $AKey.\hat{C}$;
  $text_{ct} :=$ symdec $enc_{ct}, AKey$;
  match $text_{ct}$ as $\hat{C}$;
  new $SKey$;
  $st :=$ symenc $SKey.\hat{C}, k_{S,T}^{s \to t}$;
  $enc_{tc} :=$ symenc $SKey.n_2.\hat{S}, AKey$;
  send $\hat{C}.st.enc_{tc}$;
  $]_T$

**Server** $= (S, \hat{T})$ [
  receive $st.enc_{cs}$;
  $text_{st} :=$ symdec $st, k_{S,T}^{s \to t}$;
  match $text_{st}$ as $SKey.\hat{C}$;
  $text_{cs} :=$ symdec $enc_{cs}, SKey$;
  match $text_{cs}$ as $\hat{C}.t$;
  $enc_{sc} :=$ symenc $t, SKey$;
  send $enc_{sc}$;
  $]_S$
```

Table 4.3: Roles of Kerberos with DHINIT

$$\begin{aligned}
SEC_k &: \mathsf{Hon}(\hat{C}, \hat{K}) \supset (\mathsf{GoodKeyAgainst}(X, k) \vee \hat{X} \in \{\hat{C}, \hat{K}\}) \\
SEC_{akey} &: \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset (\mathsf{GoodKeyAgainst}(X, AKey) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}) \\
SEC_{skey} &: \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}, \hat{S}) \supset (\mathsf{GoodKeyAgainst}(X, SKey) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}, \hat{S}\}) \\
AUTH_{kas} &: \exists \eta.\ \mathsf{Send}((\hat{K}, \eta), Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}).E_{sym}[k^{t \to k}_{T,K}](AKey.\hat{C}). \\
&\qquad E_{sym}[k](AKey.n_1.\hat{T})) \\
AUTH_{tgs} &: \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s \to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))
\end{aligned}$$

$SEC^{client}_{k} : [\textbf{Client}]_C\ SEC_k$	$SEC^{kas}_{k} : [\textbf{KAS}]_K\ SEC_k$
$SEC^{client}_{akey} : [\textbf{Client}]_C\ SEC_{akey}$	$AUTH^{client}_{kas} : [\textbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}) \supset AUTH_{kas}$
$SEC^{kas}_{akey} : [\textbf{KAS}]_K\ SEC_{akey}$	$AUTH^{tgs}_{kas} : [\textbf{TGS}]_T\ \mathsf{Hon}(\hat{T}, \hat{K}) \supset \exists n_1, k, gy, \tilde{n_1}.\ AUTH_{kas}$
$SEC^{tgs}_{akey} : [\textbf{TGS}]_T\ SEC_{akey}$	
	$AUTH^{client}_{tgs} : [\textbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset AUTH_{tgs}$
$SEC^{client}_{skey} : [\textbf{Client}]_C\ SEC_{skey}$	$AUTH^{server}_{tgs} : [\textbf{Server}]_S\ \mathsf{Hon}(\hat{S}, \hat{T}) \supset \exists n_2, AKey.\ AUTH_{tgs}$
$SEC^{tgs}_{skey} : [\textbf{TGS}]_T\ SEC_{skey}$	

Table 4.4: DHINIT Security Properties

also not necessary to establish reasonable security properties of practical protocols (see [36] for further discussion). We abbreviate the honesty assumptions by defining $\mathsf{Hon}(\hat{X}_1, \hat{X}_2, \cdots, \hat{X}_n) \equiv \mathsf{Honest}(\hat{X}_1) \wedge \mathsf{Honest}(\hat{X}_2) \wedge \cdots \mathsf{Honest}(\hat{X}_n)$.

The following protocol execution demonstrates that $AUTH^{client}_{kas}$ does *not* hold after the first stage of the client role.

$$\begin{aligned}
C \longrightarrow K(I) &: Cert_C.SIG[sk_C](\text{“}Auth\text{”}.HASH(\hat{C}.\hat{T}.n_1).\tilde{n_1}.gx).\hat{C}.\hat{T}.n_1 \\
I \longrightarrow K &: Cert_I.SIG[sk_I](\text{“}Auth\text{”}.HASH(\hat{I}.\hat{T}.n_1).\tilde{n_1}.gx).\hat{I}.\hat{T}.n_1 \\
K \longrightarrow I \longrightarrow C &: Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}). \\
&\qquad E_{sym}[k^{t \to k}_{T,K}](AKey.\hat{I}).E_{sym}[k](AKey.n_1.\hat{T})
\end{aligned}$$

C cannot parse the incorrect tgt : $E_{sym}[k^{t \to k}_{T,K}](AKey.\hat{I})$, as it does not have the key $k^{t \to k}_{T,K}$. Consequently, after interacting with the KAS the client is not guaranteed that the KAS thinks it interacted with the client. This problem can be easily fixed by requiring the KAS to include the client's identity inside the signature. However, the subsequent interaction with the TGS does ensure that the KAS indeed intended communication with the given client.

Theorem 21 (KAS Authentication) *On execution of the* **Client** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended KAS indeed sent the expected response assuming that both the client and the KAS are honest, the signature scheme is CMA-secure, the encryption scheme is IND-CCA and INT-CTXT secure, and the Decisional Diffie-Hellman (DDH) assumption holds. A similar result also holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash AUTH_{kas}^{client}, AUTH_{kas}^{tgs}$.

The axiomatic proof is in Appendix B.1. The key steps of the proof are the following: (a) the client C verifies the KAS K's signature on its Diffie-Hellman public value (gy) and the client's nonce ($\tilde{n_1}$) and infers using the **SIG** axiom that the KAS did produce the signature; (b) a program invariant (proved using the honesty rule **HON**) is used to infer that the KAS observed the client's nonce and produced the DH exponential gy by exponentiating some nonce y; (c) the next few proof steps establish that the Diffie-Hellman key k can be used as an encryption key only by C and K by proving that $\mathsf{DHSecretive}(\mathsf{X}, \mathsf{Y}, \mathsf{k})$ holds and then using the axiom **CTXG**; note that this step requires the use of the random oracle model since the soundness of **CTXG** depends on that; (d) since the client decrypted the ciphertext $E_{sym}[k](AKey.n_1.\hat{T})$ and the client did not produce it itself, we therefore infer that it must have been produced by the KAS. At this point, we are assured that the KAS agrees on $\hat{T}, gx, n$ and $AKey$. However, it still does not agree on the identity of the client. It turns out, as we will see in Theorem 22, that this partial authentication is sufficient to prove the secrecy of the authentication key ($AKey$) from the client's perspective. Now, stronger authentication properties are proved from the second stage of the protocol once the client decrypts the message $E_{sym}[AKey]$ $(SKey.n_2.\hat{S})$. We infer that some thread of $\hat{C}, \hat{K}$ or $\hat{T}$ must have produced the encryption because of ciphertext integrity. Using an invariant to reason about the special form of this ciphertext, we conclude that the encrypting thread must have received a tgt containing $AKey$ and meant for itself. Since we have proved the secrecy of $AKey$ already under the keys k and $k_{T,K}^{t \to k}$, we infer that this tgt must be keyed with one of k and $k^{t\to k}_{T,K}$ the holders of which—$\hat{C}$, $\hat{T}$ and $\hat{K}$—are honest. This reasoning is formally captured in the axiom **SDEC**. Now we use the honesty rule to infer that if an honest thread encrypted this message then it must have generated $AKey$; we know that thread is K. At this point, we conclude that the TGS agrees on the identity of the KAS. The proof that the TGS agrees on the identity of the client is similar.

Theorem 22 (Authentication Key Secrecy) *On execution of the* **Client** *role by a principal, the Authentication Key is guaranteed to be good, in the sense of IND-CCA security, assuming that the client, the KAS and the TGS are all honest, the signature scheme is CMA-secure, the encryption scheme is IND-CCA and INT-CTXT secure, and the DDH assumption holds. Similar results hold for principals executing the* **KAS** *and* **TGS** *roles. Formally,* $KERBEROS \vdash SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$.

The axiomatic proof is in B.1. The main idea is to prove by induction over the steps of the protocol that $AKey$ occurs on the network only as an encryption key or as a payload protected by encryption with the Diffie-Hellman key k or the pre-shared key $k^{t\to k}_{T,K}$. Formally, this step is carried out using the secrecy induction rule IND_{GOOD}. We therefore infer that $AKey$ is good for use as an encryption key using the axiom **GK**.

Since AKey is protected by both the DH key k and the symmetric key $k^{t\to k}_{T,K}$, therefore, we had to formulate a reduction to a multi party IND-CCA game where some of the keys can be symmetric, with either pre-shared keys or those generated by DHKE in section 4.1.3. We also consider the further generalization of also considering public keys, since that didn't involve additional innovation.

We prove additional authentication and secrecy properties about the later stages of the protocol. Since the later stages of DHINIT are the same as those in basic Kerberos, we leverage the composition theorems in prior work to reuse existing proofs [66].

Theorem 23 (TGS Authentication) *On execution of the* **Client** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended TGS indeed sent the expected response assuming that the client, the KAS and the TGS* *are all honest, the signature scheme is CMA-secure, the encryption scheme is IND-CCA and INT-CTXT secure, and the DDH assumption holds. Similar result holds for a principal executing the* **Server** *role. Formally, KERBEROS* $\vdash AUTH_{tgs}^{client}$, $AUTH_{tgs}^{server}$.

Proof Sketch. The proof of $AUTH_{tgs}^{server}$ can be instantiated from the template proof used for theorem 21 and is formally done in Appendix B.1.2. The proof of $AUTH_{tgs}^{client}$ depends on the 'goodkey'-ness of $AKey$ established by theorem 22 and is much more involved. A formal proof is given in Appendix A.1.3.

Theorem 24 (Service Key Secrecy) *On execution of the* **Client** *role by a principal, the Service Key is guaranteed to be good, in the sense of IND-CCA security, assuming that the client, the KAS, the TGS and the application server are all honest, the signature scheme is CMA-secure, the encryption scheme is IND-CCA and INT-CTXT secure, and the DDH assumption holds. Similar result holds for a principal executing the* **TGS** *role. Formally, KERBEROS* $\vdash SEC_{skey}^{client}, SEC_{skey}^{tgs}$.

Proof Sketch. The idea here is that the Service Key $SKey$ is protected by level-0 key $k_{S,T}^{s \to t}$ and level-1 key $AKey$. The proof of 'Secretive'-ness proceeds along the same line as for theorem 22 and uses derivations from theorem 23. Then we invoke axiom **GK** for level-2 keys to establish *KERBEROS* $\vdash SEC_{skey}^{client}, SEC_{skey}^{tgs}$. □

4.3 IKEv2

IKEv2 [23] is a complex protocol used to negotiate a security association at the beginning of an IPSec session. We consider the mode in which Diffie-Hellman exponentials are never reused and signatures are used for authentication. We formally model this mode of IKEv2 and provide the first formal proof that it satisfies computational authentication and security guarantees in the standard model. A significant difference from DHINIT is that the IKEv2 proofs do not require the random oracle model. At a high-level, this difference arises because in IKEv2 honest parties authenticate their own as well as their peer's Diffie-Hellman exponential using signatures. This enables us to prove the $\mathsf{DHStrongSecretive}(\mathsf{X}, \mathsf{Y}, \mathsf{k})$ property and use the **CTXGS** axiom in our proofs. Recall that in the DHINIT proofs we could only prove the weaker $\mathsf{DHSecretive}(\mathsf{X}, \mathsf{Y}, \mathsf{k})$ property and hence had to use the **CTXG** axiom, which is sound only in the random oracle model. However, the key derivation function needs to satisfy certain properties (based on issues identified in [29]).

4.3.1 Modeling IKEv2

IKEv2 has two roles—initiator (**Init**) and responder (**Resp**) and proceeds in two stages. We describe the protocol informally here while a formal description is given in Table 4.5. In the first stage, the initiator (A) generates a nonce (n) and a DH exponential ($gx = g^x$) and sends these along with some cryptographic suite information ($info_{i1}$) and an initiator flag bit denoted by "I" to the responder (B). The responder replies with its own nonce (m), DH exponential ($gy = g^y$) as well as some cryptographic suite information of its own ($info_{i2}$) and a responder flag bit denoted by "R".

In the second stage, the initiator generates keys sk_i and sk_r from the nonces m, n, the DH shared exponential g^{xy} and some auxilliary information. One part of each of the keys is used for encryption and the other for integrity. We model this as a single authenticated encryption operation. A signs its first message ($``I".info_{i1}.gx.n$) appended with the responder nonce (m) and a prf (prf_{pi}) of its identity $\hat{A}$ keyed with a key generated from g^{xy} and encrypts the result along with its identity ($\hat{A}$) and some auxilliary information ($info_{i2}$) with the key sk_i. A then sends out this encryption. The responder also generates the keys sk_i and sk_r from the shared information. It decrypts A's encrypted message with the key sk_i and verifies the signature of A on the first received message, its own nonce (m) and prf_{pi}. On the success of verification B signs its own first sent message ($``R".info_{r1}.gy.m$) appended with the initiator nonce (n) and a prf (prf_{pr}) of its identity $\hat{B}$ keyed with a key generated from g^{xy} and encrypts the result along with its identity ($\hat{B}$) and some auxilliary information ($info_{r2}$) with the key sk_r. It then sends out this message. On receipt of this message, A decrypts it with the key sk_r and then verifies the signature of $\hat{B}$ on the first received

```
**Init** = $(A, \hat{B})[$
        new $x$;
        new $n$;
        $gx$ := expg $x$;
        send $``I".info_{i1}.gx.n$;

        receive $info_{r1}.p.m$;
        $px$ := exp $p, x$;
        $sk_i$ := keygen-i $m, n, px, info_{i1}, info_{r1}$;
        $sk_r$ := keygen-r $m, n, px, info_{i1}, info_{r1}$;
        $prf_{pi}$ := prf-i $m, n, px, info_{i1}, info_{r1}, \hat{A}$;
        $auth_i$ := sign $``I".info_{i1}.gx.n.m.prf_{pi}, sk_A$;
        $enc_i$ := symenc $\hat{A}.auth_i.info_{i2}, sk_A$;
        send $enc_i$;

        receive $enc_r$;
        $text_r$ := symdec $enc_r, sk_r$;
        match $text_r$ as $\hat{B}.auth_r.info_{r2}$;
        verify $auth_r, ``R".info_{r1}.p.m.n.prf'_{pr}, vk_B$;
        $prf_{pr}$ := prf-r $m, n, px, info_{i1}, info_{r1}, \hat{B}$;
        match $prf'_{pr}$ as $prf_{pr}$;
        $]_A$
```

```
**Resp** = $(B)[$
        receive $info_{i1}.q.n$;
        new $y$;
        new $m$;
        $gy$ := expg $y$;
        send $``R".info_{r1}.gy.m$;

        receive $enc_i$;
        $qy$ := exp $q, y$;
        $sk_i$ := keygen-i $m, n, qy, info_{i1}, info_{r1}$;
        $sk_r$ := keygen-r $m, n, qy, info_{i1}, info_{r1}$;
        $text_i$ := symdec $enc_i, sk_i$;
        match $text_i$ as $\hat{A}.auth_i.info_{i2}$;
        verify $auth_i, ``I".info_{i1}.q.n.m.prf'_{pi}, vk_A$;
        $prf_{pi}$ := prf-i $m, n, qy, info_{i1}, info_{r1}, \hat{A}$;
        match $prf'_{pi}$ as $prf_{pi}$;
        $prf_{pr}$ := prf-r $m, n, qy, info_{i1}, info_{r1}, \hat{B}$;
        $auth_r$ := sign $``R".info_{r1}.gy.m.n.prf_{pr}, sk_B$;
        $enc_r$ := symenc $\hat{B}.auth_r.info_{r2}, sk_r$;
        send $enc_r$;
        $]_B$
```

Table 4.5: Formal Description of IKEv2 Roles.

message, A's own nonce n and prf_{pr}.

4.3.2 Security Properties and Proofs

The security properties of IKEv2, listed in Table 4.6, state that on completion of a thread executing one of the roles, the shared keys sk_i and sk_r satisfy the GoodKey property, i.e. they are suitable for use as encryption keys for an IND-CCA scheme. The authentication properties state that on completion of a thread executing either role, it is guaranteed with overwhelming probability that the intended peer indeed received and sent the corresponding messages.

Theorem 25 (IKEv2 Key Secrecy) *On execution of the* **Init** *role by a principal,*

$$
\begin{array}{rl}
SEC_{sk}^{init}: & [\textbf{Init}]_A\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \wedge \hat{X} \neq \hat{B} \wedge \hat{X} \neq \hat{B} \supset \mathsf{GoodKeyAgainst}(X, sk_i) \wedge \mathsf{GoodKeyAgainst}(X, sk_r) \\
SEC_{sk}^{resp}: & [\textbf{Resp}]_B\ \mathsf{Honest}(\hat{A}) \wedge \mathsf{Honest}(\hat{B}) \wedge \hat{X} \neq \hat{B} \wedge \hat{X} \neq \hat{B} \supset \mathsf{GoodKeyAgainst}(X, sk_i) \wedge \mathsf{GoodKeyAgainst}(X, sk_r) \\
AUTH_{resp}^{init}: & [\textbf{Init}]_A\ \exists \eta.\ B = (\hat{B}, \eta) \wedge \mathsf{Receive}(B, \text{“}I\text{”}.info_{i1}.gx.n) < \\
& \mathsf{Send}(B, \text{“}R\text{”}.info_{i2}.gy.m) < \mathsf{Receive}(B, enc_i) < \mathsf{Send}(B, enc_r) \\
AUTH_{init}^{resp}: & [\textbf{Resp}]_B\ \exists \eta.\ A = (\hat{A}, \eta) \wedge \mathsf{Send}(A, \text{“}I\text{”}.info_{i1}.gx.n) < \\
& \mathsf{Receive}(A, \text{“}R\text{”}.info_{i2}.gy.m) < \mathsf{Send}(A, enc_i)
\end{array}
$$

Table 4.6: IKEv2 Security Properties

the keys sk_i, sk_r are guaranteed to be good, in the sense of IND-CCA security, assuming that the Iniatiator and the Responder are both honest, the signature scheme is CMA-secure, the encryption scheme is IND-CCA and INT-CTXT secure, and the DDH assumption holds. Similar result holds for a principal executing the **Resp** *role. Formally, IKEv2* $\vdash SEC_{sk}^{init}, SEC_{sk}^{resp}$.

The formal proof is carried out using the proof system and is in Table 4.7. The main steps are sketched here. While executing the **Init** role, the initiator thread A verifies the signature of the responder $\hat{B}$ on the message “R”$.info_{r1}.p.m.n.prf_{pr}$. Assuming CMA-security of the signature scheme we infer that some thread B of the principal $\hat{B}$ produced this signature. Now, we prove an invariant of the protocol which lets us conclude that since B signed a message of this form, p is a DH exponential generated by B, say $p = g^y$, and that only “safe” messages constructed from this DH exponential were sent out. Assuming that the prf_{pr} could only be generated by B if it computed g^{xy}, we are assured that B received A's actual DH exponential g^x. Given this, and the fact that A itself only sent out only “safe” messages constructed from its own DH exponential g^x, we conclude that a key generated from the shared DH exponential g^{xy} should be secret, in the sense of IND-CCA security. Thus we conclude SEC_{sk}^{init}. The proof of SEC_{sk}^{resp} is similar.

Note that public information like the nonces and some auxilliary information are also used by the key generation function. The cryptographic security that we require from the key generation function is that, given $m, n, info_{i1}, info_{r1}$, the symmetric

	$[\textbf{Init}]_A\ \mathsf{Verify}(A, SIG[sk_B](\text{“}R\text{”}.info_{r1}.p.m.n.prf_{pr}), vk_B)$	(4.3)
SIG	$[\textbf{Init}]_A\ \exists \eta.\ \mathsf{Sign}((\hat{B}, \eta), \text{“}R\text{”}.info_{r1}.p.m.n.prf_{pr}, sk_B)$	(4.4)
(-1), Inst $(\hat{B}, \eta) \mapsto B$	$[\textbf{Init}]_A\ \mathsf{Sign}(B, \text{“}R\text{”}.info_{r1}.p.m.n.prf_{pr}, sk_B)$	(4.5)
HON, (-1)	$[\textbf{Init}]_A\ \exists u.\ \mathsf{New}(B, u) \wedge p = g^u \wedge \mathsf{Expg}(B, u)$	(4.6)
(-1), Inst $u \mapsto y$	$[\textbf{Init}]_A\ \mathsf{New}(B, y) \wedge p = g^y \wedge \mathsf{Expg}(B, y)$	(4.7)
(-1), **HON**, PRF uses g^{xy}	$[\textbf{Init}]_A\ \mathsf{New}(B, y) \wedge \mathsf{SendDHGood}(B, y) \wedge (\mathsf{Exp}(B, gx, y) \supset gx = g^x)$	(4.8)
		(4.9)
DH$*$, **SDH**$*$	$[\textbf{Init}]_A\ \mathsf{New}(A, x) \wedge \mathsf{SendDHGood}(A, x) \wedge (\mathsf{Exp}(A, gy, x) \supset gy = g^y)$	(4.10)
DH, $(-2, -1)$	$[\textbf{Init}]_A\ \mathsf{SharedSecret}(A, B, KeyGen_i(m, n, g^{xy}, info_{i1}, info_{r1})) \wedge \mathsf{SharedSecret}(A, B, KeyGen_r(m, n, g^{xy}, info_{i1}, info_{r1}))$	(4.11)

Table 4.7: Formal Proof of Secrecy of IKEv2 Diffie-Hellman Keys

key scheme should be IND-CCA secure with respect to the keys generated from the distribution $\{KeyGen\ (m, n, g^z, info_{i1}, info_{i2}) \mid z \xleftarrow{\$} [1, ord(g)]\}$. Also, since we have two different keys generated from the same parameters ($KeyGen_i$ and $KeyGen_r$ just take different bits from a string generated by a combination of prfs applied to the parameters), we require that the two keys be “sufficiently” independent. Following the results in [4], we believe that these properties are satisfied by a secure PRF, although we have not formally proved so. In this proof it is also assumed that given a key sk generated from a set of parameters $params$, it is computationally infeasible to find another set of parameters $params'$ such that they generate the same sk. This can be achieved by assuming that the key generation function is second-preimage resistant. All these properties can be certainly achieved by modelling the key generation functions as random oracles.

Theorem 26 (IKEv2 Authentication) *On execution of the* **Init** *role by a principal, it is guaranteed with asymptotically overwhelming probability that the intended Responder indeed received the intended messages and sent the expected responses assuming that both the Initiator and the Responder are honest, the signature scheme is CMA-secure, the encryption scheme is IND-CCA and INT-CTXT secure, and the DDH assumption holds. A similar result also holds for a principal executing the* **Resp**

role. Formally, IKEv2 $\vdash AUTH^{init}_{resp}, AUTH^{resp}_{init}$.

The authentication results are based on the signature and cryptographic properties of the keys sk_i and sk_r. By verifying the signature over "R".$info_{r1}.p.m.n.prf_{pr}$, the thread A is assured that B indeed sent the first message "R".$info_{r1}.p.m$. However, it is still not guaranteed that it received A's first message correctly. We arrive at the latter conclusion by the following chain of reasoning: 1. Since A decrypts the message $E_{sym}[sk_r](\hat{B}.auth_r.info_{r2})$ and sk_r is a shared key between the threads A and B and the encryption scheme provides ciphertext integrity (INT-CTXT), one of them must have produced the encryption. 2. We infer that B produced the encryption by establishing an invariant that an honest thread only encrypts a message with its own identity at the beginning of the message. 3. Now we reason, by establishing another invariant, that B only encrypts this message in the responder role because it earlier decrypted a message using the key sk_i. Again, since sk_i is a shared key between A and B, so either of them did the encryption. Reasoning that B in a responder's role would not encrypt a message containing a signature on a message having the initiator flag ("I"), we conclude that A produced the encryption. 4. At this point, we know that B agrees on the identity of A because it was included in A's encryption decrypted by B. Now, we know B checked A's signature on A's supposed first message. However, it is an invariant of the protocol that a single thread only signs one message. So this must be the signature on A's actual first message. This lets us conclude $AUTH^{init}_{resp}$. The proof of $AUTH^{resp}_{init}$ is similar.

(4.4) $[\textbf{Init}]_A$ $\mathsf{Sign}(B, \text{“}R\text{”}.info_{r1}.g^y.m.n.prf_{pr}, sk_B)$ (4.12)

HON $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Sign}(X, \text{“}R\text{”}.info_2.g^{y_0}.m_0.n_0.prf_{pr0}, sk_X) \supset$

$\exists info_1, q_0, enc_{i0}, enc_{r0}, prf_{pi0}.\ \mathsf{Receive}(X, \text{“}I\text{”}.info_1.q_0.n_0) <$

$\mathsf{Send}(X, \text{“}R\text{”}.info_2.g^{y_0}.m_0) < \mathsf{Receive}(X, enc_{i0}) < \mathsf{Send}(X, enc_{r0}) \wedge$

$\mathsf{New}(X, m_0) \wedge \mathsf{New}(X, y_0) \wedge$

$\exists sk_{i0}.\ \mathsf{KeyGen}_i(X, sk_{i0}, (m_0, n_0, q_0^{y_0}, info_1, info_2)) \wedge$

$\exists sk_{r0}.\ \mathsf{KeyGen}_r(X, sk_{r0}, (m_0, n_0, q_0^{y_0}, info_1, info_2)) \wedge$

$\exists \hat{Y}, auth_{i0}, info_3.\ \mathsf{SymDec}(X, enc_{i0}, sk_{i0}) \wedge enc_{i0} = E_{sym}[sk_{i0}](\hat{Y}.auth_{i0}.info_3) \wedge$

$\mathsf{Verify}(X, auth_{i0}, vk_Y) \wedge auth_{i0} = SIG[sk_Y](\text{“}I\text{”}.info_1.q_0.n_0.m_0.prf_{pi0}) \wedge$

$\exists info_4.\ \mathsf{SymEnc}(X, \hat{X}.SIG[sk_X](\text{“}R\text{”}.info_2.g^{y_0}.m_0.n_0.prf_{pr0}).info_4, sk_{r0}) \wedge$

$enc_{r0} = E_{sym}[sk_{r0}](\hat{X}.SIG[sk_X](\text{“}R\text{”}.info_2.g^{y_0}.m_0.n_0.prf_{pr0}).info_4)$ (4.13)

Inst $[\textbf{Init}]_A$ $\mathsf{Receive}(B, \text{“}I\text{”}.info_1.q_0.n) <$

$\mathsf{Send}(B, \text{“}R\text{”}.info_{r1}.g^y.m) < \mathsf{Receive}(B, enc_{i0}) < \mathsf{Send}(B, enc_{r0}) \wedge$

$\mathsf{New}(B, m) \wedge \mathsf{New}(B, y) \wedge$

$\mathsf{KeyGen}_i(B, sk_{i0}, (m, n, q_0^y, info_1, info_{r1})) \wedge$

$\mathsf{KeyGen}_r(B, sk_{r0}, (m, n, q_0^y, info_1, info_{r1})) \wedge$

$\mathsf{SymDec}(B, enc_{i0}, sk_{i0}) \wedge enc_{i0} = E_{sym}[sk_{i0}](\hat{Y}.auth_{i0}.info_3) \wedge$

$\mathsf{Verify}(B, auth_{i0}, vk_Y) \wedge auth_{i0} = SIG[sk_Y](\text{“}I\text{”}.info_1.q_0.n.m.prf_{pi}) \wedge$

$\mathsf{SymEnc}(B, \hat{B}.SIG[sk_B](\text{“}R\text{”}.info_{r1}.g^y.m.n.prf_{pr}).info_4, sk_{r0}) \wedge$

$enc_{r0} = E_{sym}[sk_{r0}](\hat{B}.SIG[sk_B](\text{“}R\text{”}.info_{r1}.g^y.m.n.prf_{pr}).info_4)$ (4.14)

$[\textbf{Init}]_A$ $\mathsf{SymDec}(A, E_{sym}[sk_r](\hat{B}.auth_r.info_{r2}), sk_r)$ (4.15)

CTXG, SEC_{sk}^{init} $[\textbf{Init}]_A$ $\mathsf{SymEnc}(B, \hat{B}.auth_r.info_{r2}, sk_r)$ (4.16)

HON $\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, \hat{Y}.auth_0.info_0, k_0) \wedge \mathsf{SymEnc}(X, \hat{Z}.auth_1.info_1, k_1) \supset$
$$\hat{Y} = \hat{Z} = \hat{X} \wedge auth_0 = auth_1 \wedge info_0 = info_1 \wedge k_0 = k_1 \tag{4.17}$$

$(-2, -1)$ $$sk_{r0} = sk_r \wedge info_4 = info_{r2} \tag{4.18}$$

Col.Res.KeyGen $$sk_{i0} = sk_i \tag{4.19}$$

CTXG $$[\mathbf{Init}]_A\ \exists X \in \{A, B\}.\ \mathsf{SymEnc}(X, \hat{Y}.auth_{i0}.info_3, sk_i) \tag{4.20}$$

Inst $X \mapsto X_0$ $$[\mathbf{Init}]_A\ \mathsf{SymEnc}(X_0, \hat{Y}.auth_{i0}.info_3, sk_i) \wedge (X_0 = A \vee X_0 = B) \tag{4.21}$$

$(4.17, -1)$ $$\hat{Y} = \hat{X_0} \tag{4.22}$$

$(-2, -1)$ $$[\mathbf{Init}]_A\ \mathsf{SymEnc}(X_0, \hat{X_0}.SIG[sk_{X_0}](\text{“}I\text{”}.info_1.q_0.n.m.prf_{pi}).info_3, sk_i) \tag{4.23}$$

$$[\mathbf{Init}]_A\ \forall msgs.\ \neg\mathsf{Sign}(B, \text{“}I\text{”}.msgs, sk_B) \tag{4.24}$$

$$[\mathbf{Init}]_A\ X_0 = A \tag{4.25}$$

HON $\mathsf{Honest}(\hat{X}) \wedge \mathsf{Sign}(X, \text{“}I\text{”}.info_0.q_0.n_0.m_0.prf_{pi0}) \wedge \mathsf{Sign}(X, \text{“}I\text{”}.info_1.q_1.n_1.m_1.prf_{pi1}) \supset$
$$info_0 = info_1 \wedge q_0 = q_1 \wedge n_0 = n_1 \wedge m_0 = m_1 \wedge prf_{pi0} = prf_{pi1} \tag{4.26}$$

$$[\mathbf{Init}]_A\ \mathsf{Sign}(A, \text{“}I\text{”}.info_{i1}.gx.n.m.prf_{pi}) \tag{4.27}$$

$(-2, -1)$ $$[\mathbf{Init}]_A\ info_1 = info_{i1} \wedge q_0 = gx \tag{4.28}$$

$$[\mathbf{Init}]_A\ \mathsf{SymEnc}(A, \hat{A}.auth_i.info_{i2}) \tag{4.29}$$

$(4.17, -1)$ $$info_3 = info_{i2} \tag{4.30}$$

$$[\mathbf{Init}]_A\ \mathsf{Receive}(B, \text{“}I\text{”}.info_{i1}.gx.n) < \mathsf{Send}(B, \text{“}R\text{”}.info_{i2}.gy.m)$$
$$< \mathsf{Receive}(B, enc_i) < \mathsf{Send}(B, enc_r) \tag{4.31}$$

Chapter 5

Conclusions

In this dissertation, we develop foundations for inductive proofs of computational security properties by proving connections between selected trace properties and useful non-trace properties. Computational secrecy properties, such as indistinguishability and suitability of a key, are not trace-based properties, making it awkward to reason inductively or compositionally about them. We therefore formulate the *secretive* trace-based property and prove that any secretive protocol can be used to construct a generic reduction from protocol attacks to attacks on underlying primitives. This allows computational secrecy to be established by direct inductive reasoning about a relatively natural and intuitive trace-based property.

A second contribution of this dissertation is a proof system for secrecy, in a formal logic based on inductive reasoning about protocol actions carried out by honest parties (only). We illustrate the power of this system by giving a modular, formal proof of computational authentication and secrecy properties of the Kerberos V5 protocol, thus addressing an open problem posed in [6]. Other proofs have been carried out, such as for a protocol that poses a challenge for the rank function method [38], but are omitted here.

Thirdly, we develop axioms and rules for proving authentication and secrecy properties of protocols that use Diffie-Hellman key exchange in combination with other mechanisms. The resulting reasoning method, which reflects intuitive informal direct arguments, is proved computationally sound by showing the existence of conventional cryptographic reductions. We prove security of Kerberos with DHINIT, as defined in the RFC [71], in the *random oracle model,* and prove security in the standard model for a modification in which the KAS signs both the Diffie-Hellman exponentials. We also discover that the KAS is *not* authenticated to the client after the first stage and suggest a fix to the protocol to avoid this problem. While IKEv2 [23] provides for several cryptographic options, we focus on the mode in which Diffie-Hellman exponentials are never reused and signatures are used for authentication. We prove that IKEv2 satisfies computational authentication and secrecy guarantees in the standard model. Intuitively, we do not need the random oracle assumption because honest IKEv2 parties authenticate both their own and their peer's Diffie-Hellman exponentials, which we believe is a prudent engineering practice.

While several exciting directions are foreseeable in network protocol security research, we are working on two promising problems in this area with interesting preliminary results. The first problem is to automate verification and generation of security proofs in PCL. We have successfully automated the verification of a large class of protocol properties using Prolog as an implementation tool and published the results recently [62]. The second problem is to provide quantitative guarantees of security for network protocols. This is a step which is a further refinement of the asymptotic guarantees provided by Computational PCL. Our approach is to quantitatively analyze individual reasoning steps in Computational PCL and combine them to arrive at a single numerical guarantee for the property being proved [37]. This is an extremely important problem to solve as this gives protocol designers a concrete idea of measures like how many times a particular key should be used. Additionally, this is a fairly unexplored territory in the formal methods community.

Appendix A

Inductive Proofs of Computational Secrecy

A.1 Proof of Kerberos Security Properties

We assume that long term symmetric keys possessed by pairs of honest principals are possessed by only themselves and are only used as keys, i.e. these are level-0 keys.

$$\Gamma_0 : \forall X, Y, Z, type.\ \mathsf{Hon}(\hat{X}, \hat{Y}) \wedge \mathsf{Possess}(Z, k_{X,Y}^{type}) \supset (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y})$$

A.1.1 Proofs of $AUTH_{kas}^{client}, AUTH_{kas}^{tgs}$ and $AUTH_{tgs}^{server}$

We first give a template proof of $[\mathbf{Role}]_X\ \mathsf{Hon}(\hat{X}, \hat{Y}) \supset \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k_{X,Y}^{type})$, where **Role** decrypts the term $E_{sym}[k_{X,Y}^{type}](M)$. Reference to equations by negative numbers is relative to the current equation - e.g. (-1) refers to the last equation. Reference by positive numbers indicates the actual number of the equation.

$$\mathsf{Hon}(\hat{X},\hat{Y}),\Gamma_0 \quad \mathsf{GoodKey}(k^{type}_{X,Y}) \tag{A.1}$$

$$\text{Hypothesis} \quad [\mathbf{Role}]_X\ \mathsf{SymDec}(X, E_{sym}[k^{type}_{X,Y}](M), k^{type}_{X,Y}) \tag{A.2}$$

$$\mathbf{CTX0}, (-2,-1) \quad [\mathbf{Role}]_X\ \exists Z.\ \mathsf{SymEnc}(Z, M, k^{type}_{X,Y}) \tag{A.3}$$

$$\text{Inst } Z \mapsto Z_0, (-1) \quad [\mathbf{Role}]_X\ \mathsf{SymEnc}(Z_0, M, k^{type}_{X,Y}) \tag{A.4}$$

$$\mathbf{A2}, (-1) \quad [\mathbf{Role}]_X\ \mathsf{Possess}(Z_0, k^{type}_{X,Y}) \tag{A.5}$$

$$\mathsf{Hon}(\hat{X},\hat{Y}),\Gamma_0,(-1) \quad [\mathbf{Role}]_X\ \hat{Z}_0 = \hat{X} \vee \hat{Z}_0 = \hat{Y} \tag{A.6}$$

$$(-4,-1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{X},\eta), M, k^{type}_{X,Y}) \vee \exists \eta.\ \mathsf{SymEnc}((\hat{Y},\eta), M, k^{type}_{X,Y}) \tag{A.7}$$

Case 1 : $\hat{X} = \hat{Y}$

$$(-1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y},\eta), M, k^{type}_{X,Y}) \tag{A.8}$$

Case 2 : $\hat{X} \neq \hat{Y}$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}_0) \wedge \hat{X}_0 \neq \hat{Y}_0 \supset \neg\mathsf{SymEnc}(X_0, M_0, k^{type}_{X_0,Y_0}) \tag{A.9}$$

$$\mathsf{Hon}(\hat{X}), (-1) \quad [\mathbf{Role}]_X\ \neg\exists \eta.\ \mathsf{SymEnc}((\hat{X},\eta), M, k^{type}_{X,Y}) \tag{A.10}$$

$$(-4,-1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y},\eta), M, k^{type}_{X,Y}) \tag{A.11}$$

Instantiating for $AUTH^{client}_{kas}$:

$$[\mathbf{Client}]_C\ \exists \eta.\ \mathsf{SymEnc}((\hat{K},\eta), (AKey, n_1, \hat{T}), k^{c\rightarrow k}_{C,K}) \tag{A.12}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, (Key, n, \hat{T}_0), k^{c\rightarrow k}_{C_0,X}) \supset \mathsf{Send}(X, \hat{C}_0, E_{sym}[k^{t\rightarrow k}_{T_0,X}](Key, \hat{C}_0), E_{sym}[k^{c\rightarrow k}_{C_0,X}](Key, n, \hat{T}_0)) \tag{A.13}$$

$$\mathsf{Hon}(\hat{K}), (-2,-1) \quad [\mathbf{Client}]_C\ \exists \eta.\ \mathsf{Send}((\hat{K},\eta), \hat{C}, E_{sym}[k^{t\rightarrow k}_{T,K}](AKey, \hat{C}), E_{sym}[k^{c\rightarrow k}_{C,K}](AKey, n_1, \hat{T})) \tag{A.14}$$

$$(-1) \quad AUTH^{client}_{kas} \tag{A.15}$$

Instantiating for $AUTH^{tgs}_{kas}$:

$$[\mathbf{TGS}]_T\ \exists\eta.\ \mathsf{SymEnc}((\hat{K},\eta),(AKey,\hat{C}),k^{t\to k}_{T,K}) \tag{A.16}$$

$$\begin{aligned} \mathbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X,(Key,\hat{C}_0),k^{t\to k}_{Y,X}) \\ & \supset \exists n.\ \mathsf{Send}(X,\hat{C}_0,E_{sym}[k^{t\to k}_{Y,X}](Key,\hat{C}_0),E_{sym}[k^{c\to k}_{C_0,X}](Key,n,\hat{Y})) \end{aligned} \tag{A.17}$$

$$\begin{aligned} \mathsf{Hon}(\hat{K}),(-2,-1) \quad & [\mathbf{TGS}]_T\ \exists\eta,n.\ \mathsf{Send}((\hat{K},\eta),\hat{C},E_{sym}[k^{t\to k}_{T,K}](AKey,\hat{C}), \\ & E_{sym}[k^{c\to k}_{C,K}](AKey,n_1,\hat{T})) \end{aligned} \tag{A.18}$$

$$(-1) \quad AUTH^{tgs}_{kas} \tag{A.19}$$

Instantiating for $AUTH^{server}_{tgs}$:

$$[\mathbf{Server}]_S\ \exists\eta.\ \mathsf{SymEnc}((\hat{T},\eta),(SKey,\hat{C}),k^{s\to t}_{S,T}) \tag{A.20}$$

$$\begin{aligned} \mathbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X,(Key,\hat{C}_0),k^{s\to t}_{Y,X}) \\ & \supset \exists n,Key'.\ \mathsf{Send}(X,\hat{C}_0,E_{sym}[k^{s\to t}_{Y,X}](Key,\hat{C}_0),E_{sym}[Key'](Key,n,\hat{Y})) \end{aligned} \tag{A.21}$$

$$\begin{aligned} \mathsf{Hon}(\hat{T}),(-2,-1) \quad & [\mathbf{Server}]_S\ \exists\eta,n,Key'.\ \mathsf{Send}((\hat{T},\eta),\hat{C},E_{sym}[k^{s\to t}_{S,T}](SKey,\hat{C}), \\ & E_{sym}[Key'](SKey,n,\hat{S})) \end{aligned} \tag{A.22}$$

$$(-1) \quad AUTH^{server}_{tgs} \tag{A.23}$$

A.1.2 Proofs of $SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$

Proof Sketch. As the form of the secrecy induction suggests, we do an induction over all the basic sequences of *KERBEROS*. The variables in the basic sequences are consistently primed in the formal proof so that no unintentional variable binding occurs. Broadly, the induction uses a combination of the following types of reasoning:

- The 'good'-ness axioms (**G**$*$) enumerated in the proof system section. The structure of Kerberos suggests that in many of the basic sequences the messages being sent out are functions of messages received. A key strategy here is to use **G2** to derive that the message received is good and then proceed to prove that the messages being sent out are also constructed in a good way. Consider as an example the sequence of actions by an application server thread $[\textbf{Server}]_{S'}$: S' receives a message $E_{sym}[SKey'](\hat{C}', t')$ and sends out a message $E_{sym}[SKey'](t')$. It is provable, just by using the $\textbf{G}*$ axioms that the later message is good if the former message is good.

- Derivations from Φ: The structure of Φ is dictated by the structure of the basic sequences we are inducing over. A practical proof strategy is starting the induction without figuring out a Φ at the outset and construct parts of the Φ as we do induction over an individual basic sequence. In case of *KERBEROS*, these parts are formulae that state that the generating thread of the putative secret $AKey$ did not perform certain types of action on $AKey$ or did it in a restricted form. The motivation for this structure of the Φ parts is that many of the basic sequences generate new nonces and send them out unprotected or protected under a set of keys different from $\mathcal{K}$. The Φ parts tell us that this is not the way the secret in consideration was sent out. For example consider one of the parts $\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))$ - this tells us that the generator of $AKey$ did not send it out unprotected in the open.

- Derivations from the θ's, that is, the preconditions. These are conditions which are true at the beginning of the basic sequence we are inducing over with respect to the staged control flow that *KERBEROS* exhibits. As before, a practical proof strategy is to find out what precondition we need for the secrecy induction and do the precondition induction part afterwards. Consider for example the end of the first stage of the client thread $[\textbf{Client}]_{C'}$. We know that at the beginning of the second stage the following formula always holds - $\mathsf{Receive}(C', \hat{C}'.tgt'.enc'_{kc})$ from which we derive $\theta : \mathsf{Good}(C', tgt', AKey, \mathcal{K})$. The reason this information is necessary is that the second stage sends out tgt' in the open - in order to reason that this is good to send out we use the precondition θ.

Formal Proof. We formally prove the secrecy of the session key $AKey$ with respect to the set of keys $\mathcal{K} = \{k^{c \to k}_{C,K}, k^{t \to k}_{T,K}\}$. The assumed condition Φ is the conjunction of the following formulae:

$$\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))$$

$$\Phi_2 : \forall X, \hat{C}_0, \hat{K}_0, \hat{T}_0, n.\ \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.n.\hat{T}_0, k^{c\to k}_{C_0,K_0}) \supset \hat{X} = \hat{K} \wedge \hat{C}_0 = \hat{C} \wedge \hat{T}_0 = \hat{T}$$

$$\Phi_3 : \forall X, \hat{S}_0, \hat{C}_0.\ \mathsf{New}(X, AKey) \supset \neg\mathsf{SymEnc}(X, AKey.\hat{C}_0, k^{s\to t}_{S_0,X})$$

Observe that Φ is prefix closed.

Now we present the formal proof of goodness:

Let, $[\mathbf{Client}_1]_{C'} : [\texttt{new}\ n_1'; \texttt{send}\ \hat{C}'.\hat{T}'.n_1';]_{C'}$

	$[\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n_1') \wedge \mathsf{Send}(C', \hat{C}'.\hat{T}'.n_1')$	(A.24)
$\Phi_1, (-1)$	$[\mathbf{Client}_1]_{C'}\ n_1' \neq AKey$	(A.25)
$\mathbf{G1}, \mathbf{G5}, (-1)$	$[\mathbf{Client}_1]_{C'}\ \mathsf{Good}(C', \hat{C}'.\hat{T}'.n_1', AKey, \mathcal{K})$	(A.26)
$\mathbf{SG1\text{-}2}, (-1)$	$\mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K})$	(A.27)

Let, $[\mathbf{Client}_2]_{C'} : [\texttt{receive}\ \hat{C}'.tgt'.enc_{kc}';$
$text_{kc}' := \texttt{symdec}\ enc_{kc}', k^{c\to k}_{C',K'};$
$\texttt{match}\ text_{kc}'\ \texttt{as}\ AKey'.n_1'.\hat{T}';]_{C'}$

$\mathbf{SG1}$	$\mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K})$	(A.28)

Precondition $\theta_3 : \mathsf{Good}(C', tgt', AKey, \mathcal{K})$

Let, $[\mathbf{Client}_3]_{C'} : [\texttt{new}\ n_2';\ enc_{ct}' := \texttt{symenc}\ \hat{C}', AKey';$
$\texttt{send}\ tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2';]_{C'}$

	$[\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n_2') \wedge \mathsf{Send}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2')$	(A.29)
$\Phi_1, (-1)$	$[\mathbf{Client}_3]_{C'}\ n_2' \neq AKey$	(A.30)
$\mathbf{G1}, (-1)$	$\theta_3\ [\texttt{new}\ n_2';]_{C'}\ \mathsf{Good}(C', tgt', AKey, \mathcal{K}) \wedge \mathsf{Good}(C', n_2', AKey, \mathcal{K})$	(A.31)
$\mathbf{G}*, (-1)$	$\theta_3\ [\mathbf{Client}_3]_{C'}\ \mathsf{Good}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2', AKey, \mathcal{K})$	(A.32)
$\mathbf{SG1\text{-}2}, (-1)$	$\theta_3 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K})$	(A.33)

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ [$\texttt{receive}$ $\hat{C}', st', enc'_{tc}$;

$text'_{tc} := \texttt{symdec}$ $enc'_{tc}, AKey'$; $\texttt{match}$ $text'_{tc}$ $\texttt{as}$ $SKey'.n'_2.\hat{S}'$; $]_{C'}$

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (A.34)

Precondition θ_5 : $\mathsf{Good}(C', st', AKey, \mathcal{K})$

$\cdots$ proof for following BS similar to (13) $\cdots$

$\theta_5 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})$ [

$enc'_{cs} := \texttt{symenc}$ $\hat{C}'.t', SKey'$;

$\texttt{send}$ st', enc'_{cs}; $]_{C'}$

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (A.35)

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ [$\texttt{receive}$ enc'_{sc};

$text'_{sc} := \texttt{symdec}$ $enc'_{sc}, SKey'$; $\texttt{match}$ $text'_{sc}$ $\texttt{as}$ t'; $]_{C'}$

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (A.36)

Let, $[\mathbf{KAS}]_{K'}$: $[\texttt{receive}\ \hat{C}'.\hat{T}'.n_1';$
$\texttt{new}\ AKey';$
$tgt' := \texttt{symenc}\ AKey'.\hat{C}', k^{t\to k}_{T',K'};$
$enc'_{kc} := \texttt{symenc}\ AKey'.n_1'.\hat{T}', k^{c\to k}_{C',K'};$
$\texttt{send}\ \hat{C}'.tgt'.enc'_{kc};]_{K'}$

Case 1 : $AKey' = AKey$

$$[\mathbf{KAS}]_{K'} \mathsf{New}(K', AKey) \wedge \mathsf{SymEnc}(K', AKey.n_1'.\hat{T}', k^{c\to k}_{C',K'}) \quad (A.37)$$

$\Phi_2, (-1)$ $$[\mathbf{KAS}]_{K'} \hat{C}' = \hat{C} \wedge \hat{K}' = \hat{K} \wedge \hat{T}' = \hat{T} \quad (A.38)$$

(-1) $$[\mathbf{KAS}]_{K'} k^{c\to k}_{C',K'} \in \mathcal{K} \wedge k^{t\to k}_{T',K'} \in \mathcal{K} \quad (A.39)$$

$\mathbf{G}*, (-1)$ $$[\mathbf{KAS}]_{K'}\ \mathsf{Good}(K', \hat{C}'.tgt'.enc'_{kc}, AKey, \mathcal{K}) \quad (A.40)$$

Case 2 : $AKey' \neq AKey$

$\mathbf{G2}$ $$[\texttt{receive}\ \hat{C}'.\hat{T}'.n_1';]_{K'}\ \mathsf{Good}(K', \hat{C}'.\hat{T}'.n_1', AKey, \mathcal{K}) \quad (A.41)$$

$\mathbf{G6}, (-1)$ $$[\texttt{receive}\ \hat{C}'.\hat{T}'.n_1';]_{K'}\ \mathsf{Good}(K', n_1', AKey, \mathcal{K}) \quad (A.42)$$

$\mathbf{G}*, (-1)$ $$[\mathbf{KAS}]_{K'}\ \mathsf{Good}(\hat{C}'.tgt'.enc'_{kc}, AKey, \mathcal{K}) \quad (A.43)$$

$\mathbf{SG1\text{-}2}, (-4, -1)$ $$\mathsf{SendGood}(K', AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SendGood}(K', AKey, \mathcal{K}) \quad (A.44)$$

Let, $[\mathbf{TGS}]_{T'}$: $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n_2';$
$text'_{ct1} := \texttt{symdec}\ enc'_{ct1}, k^{t\to k}_{T',K'};$
$\texttt{match}\ text'_{ct1}\ \texttt{as}\ AKey'.\hat{C}';$
$text'_{ct2} := \texttt{symdec}\ enc'_{ct2}, AKey';$
$\texttt{match}\ text'_{ct2}\ \texttt{as}\ \hat{C}';$
$\texttt{new}\ SKey';$
$st' := \texttt{symenc}\ SKey'.\hat{C}', k^{s\to t}_{S',T'};$
$enc'_{tc} := \texttt{symenc}\ SKey'.n_2'.\hat{S}', AKey';$
$\texttt{send}\ \hat{C}'.st'.enc'_{tc};]_{T'}$

$\mathbf{G2}, \mathbf{G6}$ $$[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n_2';]_{T'}\ \mathsf{Good}(T', n_2', AKey, \mathcal{K}) \quad (A.45)$$

$$[\mathbf{TGS}]_{T'}\ \mathsf{New}(T', SKey') \wedge \mathsf{SymEnc}(T', SKey'.\hat{C}', k^{s\to t}_{S',T'}) \quad (A.46)$$

$\Phi_3, (-1)$ $$[\mathbf{TGS}]_{T'}\ SKey' \neq AKey \quad (A.47)$$

$\mathbf{G1}, (-1)$ $$[\cdots; \texttt{new}\ SKey';]_{T'}\ \mathsf{Good}(T', SKey', AKey, \mathcal{K}) \quad (A.48)$$

$\mathbf{G}*, (-4, -1)$ $$[\mathbf{TGS}]_{T'}\ \mathsf{Good}(T', \hat{C}'.st'.enc'_{tc}, AKey, \mathcal{K}) \quad (A.49)$$

$\mathbf{SG1\text{-}2}, (-1)$ $$\mathsf{SendGood}(T', AKey, \mathcal{K})\ [\mathbf{TGS}]_{T'}\ \mathsf{SendGood}(T', AKey, \mathcal{K}) \quad (A.50)$$

$$\begin{aligned}
\text{Let, } [\textbf{Server}]_{S'} : &[\texttt{receive } enc'_{cs1}.enc'_{cs2};\\
&text'_{cs1} := \texttt{symdec } enc'_{cs1}, k^{s\to t}_{S',T'}; \texttt{match } text'_{cs1} \texttt{ as } SKey'.\hat{C}';\\
&text'_{cs2} := \texttt{symdec } enc'_{cs2}, SKey'; \texttt{match } enc'_{cs2} \texttt{ as } \hat{C}'.t';\\
&enc'_{sc} := \texttt{symenc } t', SKey';\\
&\texttt{send } enc'_{sc};]_{S'}
\end{aligned}$$

$$\begin{aligned}
\text{Case 1:}\quad & SKey' \in \mathcal{K}\\
\mathbf{G7}, (-1)\quad & [\cdots; enc'_{sc} := \texttt{symenc } t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K}) && \text{(A.51)}
\end{aligned}$$

$$\begin{aligned}
\text{Case 2:}\quad & SKey' \notin \mathcal{K} && \text{(A.52)}\\
\mathbf{G2}, \mathbf{G6}\quad & [\texttt{receive } enc'_{cs1}.enc'_{cs2};]_{S'}\ \mathsf{Good}(S', enc'_{cs2}, AKey, \mathcal{K}) && \text{(A.53)}\\
\mathbf{G6}, \mathbf{G8}, (-1)\quad & [\cdots; text'_{cs2} := \texttt{symdec } enc'_{cs2}, SKey'; \texttt{match } enc'_{cs2} \texttt{ as } \hat{C}'.t';]_{S'}\\
& \mathsf{Good}(S', t', AKey, \mathcal{K}) && \text{(A.54)}\\
\mathbf{G7}, (-1)\quad & [\cdots; enc'_{sc} := \texttt{symenc } t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K}) && \text{(A.55)}\\
(-4, -1)\quad & [\textbf{Server}]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K}) && \text{(A.56)}\\
\textbf{SG1-2}, (-1)\quad & \mathsf{SendGood}(S', AKey, \mathcal{K})\ [\textbf{Server}]_{S'}\ \mathsf{SendGood}(S', AKey, \mathcal{K}) && \text{(A.57)}
\end{aligned}$$

$$\text{Theorem 12}\quad \Phi \supset \mathsf{Secretive}(AKey, \mathcal{K}) \qquad \text{(A.58)}$$

We can derive from $AUTH^{client}_{kas}$, the actions in $[\textbf{KAS}]_K$ and $AUTH^{client}_{tgs}$ that:

$$\begin{aligned}
&\mathit{KERBEROS} \vdash [\textbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi\\
&\mathit{KERBEROS} \vdash [\textbf{KAS}]_K\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi\\
&\mathit{KERBEROS} \vdash [\textbf{TGS}]_T\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi
\end{aligned}$$

The only principals having access to a key in $\mathcal{K}$ are $\hat{C}, \hat{K}$ and $\hat{T}$. All keys in $\mathcal{K}$ are level-0 as they are used only as keys. In addition, Φ_2 assumes that some thread of K generated $AKey$. Therefore, we have:

$$\begin{aligned}
\mathsf{InInitSet}(X, AKey, \mathcal{K}) &\equiv \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}\\
\mathsf{GoodInit}(AKey, \mathcal{K}) &\equiv \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T})\\
\mathsf{GoodKeyFor}(AKey, \mathcal{K}) &\equiv \mathsf{GoodKeyAgainst}(AKey, X) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}
\end{aligned}$$

Therefore, by axiom **GK**, we have:

$$\mathsf{Secretive}(AKey, \mathcal{K}) \wedge \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \Rightarrow \mathsf{GoodKeyAgainst}(X, AKey) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}$$

Combining everything we have:

$$KERBEROS \vdash SEC_{akey}^{client}, SEC_{akey}^{kas}, SEC_{akey}^{tgs}$$

A.1.3 Proof of $AUTH_{tgs}^{client}$

This proof uses the secrecy property SEC_{akey}^{client} which established the secrecy of $AKey$ among $\hat{C}, \hat{K}$ and $\hat{T}$ assuming their honesty. At a high level, the client reasons that since $AKey$ is known only to $\hat{C}, \hat{K}$ and $\hat{T}$, the term $E_{sym}[AKey](SKey.n_2.\hat{S})$ could only have been computed by one of them. Some non-trivial technical effort is required to prove that this encryption was indeed done by a thread of $\hat{T}$ and not by any thread of $\hat{C}$ or $\hat{K}$, which could have been the case if *e.g.*, there existed a reflection attack. After showing that it was indeed a thread of $\hat{T}$ who encrypted the term, we use the honesty rule to show that it indeed sent the expected response to C's message.

Again, reference to equations by negative numbers is relative to the current equation - *e.g.*, (-1) refers to the last equation. Reference by positive number indicates the actual number of the equation.

$$[\textbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[AKey](SKey.n_2.\hat{S}), AKey) \tag{A.59}$$

$$\textbf{CTXL}, (-1, 1) \quad [\textbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, SKey.n_2.\hat{S}, AKey) \tag{A.60}$$

$$\text{Inst } X \mapsto X_0, (-1) \quad [\textbf{Client}]_C\ \mathsf{SymEnc}(X_0, SKey.n_2.\hat{S}, AKey) \tag{A.61}$$

$$(-1) \quad [\textbf{Client}]_C\ \mathsf{Possess}(X_0, AKey) \tag{A.62}$$

$$SEC^{client}_{AKey}, (-1) \quad \hat{X}_0 = \hat{C} \wedge \hat{X}_0 = \hat{K} \wedge \hat{X}_0 = \hat{T} \tag{A.63}$$

$$\begin{aligned} \textbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key'.n.\hat{S}_0, Key) \wedge Key \neq k^{c\to k}_{Z,X} \\ & \supset \exists \hat{K}_0, \hat{C}_0.\ \mathsf{SymDec}(X, E_{sym}[k^{t\to k}_{X,K_0}](Key.\hat{C}_0), k^{t\to k}_{X,K_0}) \\ & \wedge \mathsf{Send}(X, \hat{C}_0.E_{sym}[k^{s\to t}_{S_0,X}](Key'.\hat{C}_0).E_{sym}[Key](Key'.n.\hat{S}_0)) \end{aligned} \tag{A.64}$$

$$\begin{aligned} \text{Inst}, (-4, -1) \quad & [\textbf{Client}]_C\ \mathsf{SymDec}(X_0, E_{sym}[k^{t\to k}_{X_0,K_0}](AKey.\hat{C}_0), k^{t\to k}_{X_0,K_0}) \\ & \wedge \mathsf{Send}(X_0, \hat{C}_0.E_{sym}[k^{s\to t}_{S,X_0}](SKey.\hat{C}_0).E_{sym}[AKey](SKey.n_2.\hat{S})) \end{aligned} \tag{A.65}$$

$$\textbf{CTX0}, (-1) \quad [\textbf{Client}]_C\ \exists Y.\ \mathsf{SymEnc}(Y, AKey.\hat{C}_0, k^{t\to k}_{X_0,K_0}) \tag{A.66}$$

$$\text{Inst } Y \mapsto Y_0, (-1) \quad [\textbf{Client}]_C\ \mathsf{SymEnc}(Y_0, AKey.\hat{C}_0, k^{t\to k}_{X_0,K_0}) \tag{A.67}$$

$$\textbf{A2}, (-1) \quad [\textbf{Client}]_C\ \mathsf{Possess}(Y_0, AKey) \tag{A.68}$$

$$SEC^{client}_{AKey}, (-1) \quad \mathsf{Honest}(\hat{Y_0}) \tag{A.69}$$

$$\textbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{W}, k^{t\to k}_{X,Z}) \supset \mathsf{New}(X, Key) \tag{A.70}$$

$$(-4, -1) \quad [\textbf{Client}]_C\ \mathsf{New}(Y_0, AKey) \tag{A.71}$$

$$\begin{aligned} AUTH^{client}_{kas} \quad & \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.\hat{W}, k^{t\to k}_{Y,Z}) \\ & \supset \hat{Y} = \hat{T} \wedge \hat{Z} = \hat{K} \wedge \hat{W} = \hat{C} \end{aligned} \tag{A.72}$$

$$(9, -2, -1) \quad \hat{X}_0 = \hat{T} \wedge \hat{K}_0 = \hat{K} \wedge \hat{C}_0 = \hat{C} \tag{A.73}$$

$$(A.65, -1) \quad [\textbf{Client}]_C\ \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s\to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S})) \tag{A.74}$$

A.2 Kerberos with PKINIT

A.2.1 Formal Description

```
**Client** = $(C, \hat{K}, \hat{T}, \hat{S}, t)$ [
  new $n_1$; new $\tilde{n_1}$;
  $sigc$ := sign $t_C.\tilde{n_1}, sk_C$;
  send $Cert_C.sigc.\hat{C}.\hat{T}.n_1$;

  receive $encp_{kc}.\hat{C}.tgt.enc_{kc}$;
  $textp_{kc}$ := pkdec $encp_{kc}, dk_C$;
  match $textp_{kc}$ as $Cert_K.sigk$;
  verify $sigk, k.ck, vk_K$;
  $\tilde{ck}$ := hash $Cert_C.sigc.\hat{C}.\hat{T}.n_1, k$;
  match $\tilde{ck}$ as $ck$;
  $text_{kc}$ := symdec $enc_{kc}, k$;
  match $text_{kc}$ as $AKey.n_1.t_K.\hat{T}$;

  $\cdots$ stage boundary $\cdots$

  new $n_2$;
  $enc_{ct}$ := symenc $\hat{C}, AKey$;
  send $tgt.enc_{ct}.\hat{C}.\hat{S}, n_2$;

  receive $\hat{C}.st.enc_{tc}$;
  $text_{tc}$ := symdec $enc_{tc}, AKey$;
  match $text_{tc}$ as $SKey.n_2.\hat{S}$;

  $\cdots$ stage boundary $\cdots$

  $enc_{cs}$ := symenc $\hat{C}.t, SKey$;
  send $st.enc_{cs}$;

  receive $enc_{sc}$;
  $text_{sc}$ := symdec $enc_{sc}, SKey$;
  match $text_{sc}$ as $t$;
  $]_C$

**KAS** = $(K)$ [
  receive $Cert_C.sigc.\hat{C}.\hat{T}.n_1$;
  verify $sigc, t_C.\tilde{n_1}, vk_C$;
  new $k$; new $AKey$;
  $ck$ := hash $Cert_C.sigc.\hat{C}.\hat{T}.n_1, k$;
  $sigk$ := sign $k.ck, sk_K$;
  $encp_{kc}$ := pkenc $Cert_K.sigk, pk_C$;
  $tgt$ := symenc $AKey.\hat{C}, k_{T,K}^{t \to k}$;
  $enc_{kc}$ := symenc $AKey.n_1.t_K.\hat{T}, k$;
  send $encp_{kc}.\hat{C}.tgt.enc_{kc}$;
  $]_K$

**TGS** = $(T, \hat{K})$ [
  receive $tgt.enc_{ct}.\hat{C}.\hat{S}.n_2$;
  $text_{tgt}$ := symdec $tgt, k_{T,K}^{t \to k}$;
  match $text_{tgt}$ as $AKey.\hat{C}$;
  $text_{ct}$ := symdec $enc_{ct}, AKey$;
  match $text_{ct}$ as $\hat{C}$;
  new $SKey$;
  $st$ := symenc $SKey.\hat{C}, k_{S,T}^{s \to t}$;
  $enc_{tc}$ := symenc $SKey.n_2.\hat{S}, AKey$;
  send $\hat{C}.st.enc_{tc}$;
  $]_T$

**Server** = $(S, \hat{T})$ [
  receive $st.enc_{cs}$;
  $text_{st}$ := symdec $st, k_{S,T}^{s \to t}$;
  match $text_{st}$ as $SKey.\hat{C}$;
  $text_{cs}$ := symdec $enc_{cs}, SKey$;
  match $text_{cs}$ as $\hat{C}.t$;
  $enc_{sc}$ := symenc $t, SKey$;
  send $enc_{sc}$;
  $]_S$
```

A.2.2 Proofs of PKINIT Security Properties

Long term symmetric keys possessed by pairs of honest principals are possessed by only themselves and are only used as keys, i.e. these are level-0 keys.

$$\Gamma_0 : \forall X, Y, Z, type.\ \mathsf{Hon}(\hat{X}, \hat{Y}) \wedge \mathsf{Possess}(Z, k_{X,Y}^{type}) \supset (\hat{Z} = \hat{X} \vee \hat{Z} = \hat{Y})$$

We will additionally use the following axiom based on collision resistance of keyed hash functions:

$$\mathbf{A3} \quad \mathsf{New}(X, k) \wedge HASH[k](m) = HASH[k](m') \supset m = m'$$

A.2.3 Proof of $AUTH_{kas}^{client}$

$$[\mathbf{Client}]_C\ \mathsf{Verify}(C, SIG[sk_K](k.ck), vk_K) \quad (A.75)$$

$$(-1) \quad [\mathbf{Client}]_C\ \exists\eta.\ \mathsf{Sign}((\hat{K}, \eta), k.ck, sk_K) \quad (A.76)$$

$$\text{Inst } (\hat{K}, \eta) \mapsto K \quad [\mathbf{Client}]_C\ \mathsf{Sign}(K, k.ck, sk_K) \quad (A.77)$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{Sign}(X, n.m, sk_X) \wedge \neg\mathsf{Time}(n) \supset \mathsf{New}(X, n) \wedge$$
$$\exists m'.\ m = HASH[k](m') \wedge \mathsf{Hash}(X, m', k) \quad (A.78)$$

$$(-2, -1) \quad [\mathbf{Client}]_C\ \mathsf{New}(K, k) \wedge \exists m'.\ ck = HASH[k](m') \wedge \mathsf{Hash}(K, m', k) \quad (A.79)$$

$$[\mathbf{Client}]_C\ ck = HASH[k](Cert_C.sigc.\hat{C}.\hat{T}.n_1) \quad (A.80)$$

$$\mathbf{A3}, (-2, -1) \quad [\mathbf{Client}]_C\ m' = Cert_C.sigc.\hat{C}.\hat{T}.n_1 \quad (A.81)$$

$$(-3, -1) \quad [\mathbf{Client}]_C\ \mathsf{New}(K, k) \wedge \mathsf{Hash}(K, Cert_C.sigc.\hat{C}.\hat{T}.n_1, k) \quad (A.82)$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{Hash}(X, Cert_{C_0}.sigc_0.\hat{C}_0.\hat{T}_0.n, k_0) \supset$$
$$\exists t, Key.\ \mathsf{Send}(X, E_{pk}[pk_{C_0}](Cert_X.SIG[sk_X](k_0.$$
$$HASH[k_0](Cert_{C_0}.sigc_0.\hat{C}_0.\hat{T}_0.n)).\hat{C}_0.E_{sym}[k_{T_0,X}^{t \to k}](Key.\hat{C}_0).$$
$$E_{sym}[k_0](Key.n.t.\hat{T}_0))) \quad (A.83)$$

$$(-2, -1) \quad [\mathbf{Client}]_C\ \exists t, Key.\ \mathsf{Send}(K, E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)).\hat{C}.$$
$$E_{sym}[k_{T,K}^{t \to k}](Key.\hat{C}).E_{sym}[k](Key.n_1.t.\hat{T})) \quad (A.84)$$

The last equation establishes $\tilde{AUTH}_{client}^{kas}$: $[\mathbf{Client}]_C\ \exists t_K, AKey.\ AUTH_{kas}$. In section A.2.5, We will use $\tilde{AUTH}_{client}^{kas}$ to prove $[\mathbf{Client}]_C\ \mathsf{Secretive}(k, \{dk_C\})$ and $\mathsf{GoodInit}(k, \{dk_C\}) \equiv \mathsf{Hon}(\hat{C}, \hat{K})$. Therefore, using axiom

CTXL we can derive:

$$\textbf{CTXL} \quad [\textbf{Client}]_C\ \mathsf{SymDec}(X, E_{sym}[k](m), k) \Rightarrow \exists Z.\ \mathsf{SymEnc}(Z, m, k) \tag{A.85}$$

Now we proceed to prove $AUTH^{kas}_{client}$:

$$[\textbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[k](AKey.n_1.t_K.\hat{T}), k) \tag{A.86}$$

$$(A.85, -1) \quad [\textbf{Client}]_C\ \exists Z_0.\ \mathsf{SymEnc}(Z_0, AKey.n_1.t_K.\hat{T}, k) \tag{A.87}$$

$$\text{Inst } Z_0 \mapsto Z, (-1) \quad [\textbf{Client}]_C\ \mathsf{SymEnc}(Z, AKey.n_1.t_K.\hat{T}, k) \tag{A.88}$$

$$\textbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.n.t.\hat{T_0}, k_0) \supset \mathsf{New}(X, k_0) \tag{A.89}$$

$$(-2, -1) \quad [\textbf{Client}]_C\ \mathsf{New}(Z, k) \tag{A.90}$$

$$\textbf{A1}, (-1) \quad [\textbf{Client}]_C\ Z = K \tag{A.91}$$

$$\textbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{Hash}(X, Cert_{C_0}.sigc_0.\hat{C_0}.\hat{T_0}.n, k_0) \supset \exists t, Key.\ \mathsf{SymEnc}(X, Key.n.t.\hat{T_0}, k_0) \tag{A.92}$$

$$(-1) \quad [\textbf{Client}]_C\ \exists t, Key.\ \mathsf{SymEnc}(K, Key.n_1.t.\hat{T}, k) \tag{A.93}$$

$$\textbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.n.t.\hat{T_0}, k_0) \wedge \mathsf{SymEnc}(X, Key'.n'.t'.\hat{T}'_0, k_0) \supset Key = Key' \wedge n = n' \wedge t = t' \wedge \hat{T_0} = \hat{T}'_0 \tag{A.94}$$

$$\tilde{AUTH}^{client}_{kas}, (-1) \quad [\textbf{Client}]_C\ \mathsf{Send}(K, E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)).\hat{C}.E_{sym}[k^{t\to k}_{T,K}](Key.\hat{C}).E_{sym}[k](AKey.n_1.t_K.\hat{T})) \tag{A.95}$$

$$(-1) \quad AUTH^{client}_{kas} \tag{A.96}$$

A.2.4 Proofs of $AUTH^{tgs}_{kas}$ and $AUTH^{server}_{tgs}$

We first give a template proof of $[\textbf{Role}]_X\ \mathsf{Hon}(\hat{X}, \hat{Y}) \supset \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y})$, where **Role** decrypts the term $E_{sym}[k^{type}_{X,Y}](M)$. Reference to equations by negative numbers is relative to the current equation - e.g. (-1) refers to the last equation. Reference by positive numbers indicates the actual number of the equation.

$$\begin{array}{rll}
\mathsf{Hon}(\hat{X},\hat{Y}),\Gamma_0 & \mathsf{GoodKey}(k^{type}_{X,Y}) & (A.97)\\
\text{Hypothesis} & [\mathbf{Role}]_X\ \mathsf{SymDec}(X, E_{sym}[k^{type}_{X,Y}](M), k^{type}_{X,Y}) & (A.98)\\
\mathbf{CTX0},(-2,-1) & [\mathbf{Role}]_X\ \exists Z.\ \mathsf{SymEnc}(Z, M, k^{type}_{X,Y}) & (A.99)\\
\text{Inst } Z\mapsto Z_0,(-1) & [\mathbf{Role}]_X\ \mathsf{SymEnc}(Z_0, M, k^{type}_{X,Y}) & (A.100)\\
\mathbf{A2},(-1) & [\mathbf{Role}]_X\ \mathsf{Possess}(Z_0, k^{type}_{X,Y}) & (A.101)\\
\mathsf{Hon}(\hat{X},\hat{Y}),\Gamma_0,(-1) & [\mathbf{Role}]_X\ \hat{Z_0}=\hat{X}\vee\hat{Z_0}=\hat{Y} & (A.102)\\
(-4,-1) & [\mathbf{Role}]_X\ \exists\eta.\ \mathsf{SymEnc}((\hat{X},\eta), M, k^{type}_{X,Y}) & \\
 & \vee\ \exists\eta.\ \mathsf{SymEnc}((\hat{Y},\eta), M, k^{type}_{X,Y}) & (A.103)
\end{array}$$

Case 1 : $\hat{X}=\hat{Y}$

$$\begin{array}{rll}
(-1) & [\mathbf{Role}]_X\ \exists\eta.\ \mathsf{SymEnc}((\hat{Y},\eta), M, k^{type}_{X,Y}) & (A.104)
\end{array}$$

Case 2 : $\hat{X}\neq\hat{Y}$

$$\begin{array}{rll}
\mathbf{HON} & \mathsf{Honest}(\hat{X_0})\wedge\hat{X_0}\neq\hat{Y_0}\supset\neg\mathsf{SymEnc}(X_0, M_0, k^{type}_{X_0,Y_0}) & (A.105)
\end{array}$$

$$\begin{array}{rll}
\mathsf{Hon}(\hat{X}),(-1) & [\mathbf{Role}]_X\ \neg\exists\eta.\ \mathsf{SymEnc}((\hat{X},\eta), M, k^{type}_{X,Y}) & (A.106)\\
(-4,-1) & [\mathbf{Role}]_X\ \exists\eta.\ \mathsf{SymEnc}((\hat{Y},\eta), M, k^{type}_{X,Y}) & (A.107)
\end{array}$$

Instantiating for $AUTH^{tgs}_{kas}$:

$$[\mathbf{TGS}]_T\ \exists\eta.\ \mathsf{SymEnc}((\hat{K},\eta),(AKey,\hat{C}),k^{t\to k}_{T,K}) \qquad (A.108)$$

$$\begin{array}{rll}
\mathbf{HON} & \mathsf{Honest}(\hat{X})\wedge\mathsf{SymEnc}(X, Key.\hat{C_0}, k^{t\to k}_{Y,X}) & \\
 & \supset\exists k_0, ck_0, n, t.\ \mathsf{Send}(X, E_{pk}[pk_{C_0}](Cert_X.SIG[sk_X](k_0.ck_0)). & \\
 & \quad \hat{C_0}.E_{sym}[k^{t\to k}_{Y,X}](Key.\hat{C_0}).E_{sym}[k_0](Key.n.t.\hat{Y})) & (A.109)
\end{array}$$

$$\begin{array}{rll}
\mathsf{Hon}(\hat{K}),(-2,-1) & [\mathbf{TGS}]_T\ \exists\eta, n_1, k, ck, t_K.\ \mathsf{Send}((\hat{K},\eta), E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)). & \\
 & \quad \hat{C}.E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C}).E_{sym}[k](AKey.n_1.t_K.\hat{T})) & (A.110)\\
(-1) & AUTH^{tgs}_{kas} & (A.111)
\end{array}$$

Instantiating for $AUTH_{tgs}^{server}$:

$$[\mathbf{Server}]_S \ \exists \eta.\ \mathsf{SymEnc}((\hat{T}, \eta), SKey.\hat{C}, k_{S,T}^{s \to t}) \tag{A.112}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C}_0, k_{Y,X}^{s \to t})$$
$$\supset \exists n, Key'.\ \mathsf{Send}(X, \hat{C}_0.E_{sym}[k_{Y,X}^{s \to t}](Key.\hat{C}_0).E_{sym}[Key'](Key.n.\hat{Y})) \tag{A.113}$$

$$\mathsf{Hon}(\hat{T}), (-2, -1) \quad [\mathbf{Server}]_S \ \exists \eta, n, Key'.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k_{S,T}^{s \to t}](SKey.\hat{C}).$$
$$E_{sym}[Key'](SKey.n.\hat{S})) \tag{A.114}$$
$$(-1) \quad AUTH_{tgs}^{server} \tag{A.115}$$

A.2.5 Proofs of $SEC_k^{client}, SEC_k^{kas}$

We will skip the detailed proof here as it is similar in structure to the subsequent section A.2.6. The assumed condition Φ is the conjunction of the following formulae:

$\Phi_0 : \forall X.\ \mathsf{New}(X, k) \supset \hat{X} = \hat{K}$

$\Phi_1 : \forall X, M.\ \mathsf{New}(X, k) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, k))$

$\Phi_2 : \forall X, t.\ \mathsf{New}(X, k) \supset \neg\mathsf{Sign}(X, t.k, sk_X)$

$\Phi_3 : \forall X, Y, ck_0.\ \mathsf{New}(X, k) \wedge \mathsf{PkEnc}(X, Cert_X.SIG[sk_X](k.ck_0), pk_Y) \supset \hat{Y} = \hat{C}$

$\Phi_4 : \forall X, m, key.\ \mathsf{New}(X, k) \supset \neg\mathsf{SymEnc}(X, k.m, key)$

A.2.6 Proofs of $SEC_{akey}^{client}, SEC_{akey}^{kas}, SEC_{akey}^{tgs}$

In this section we formally prove the secrecy of the session key $AKey$ with respect to the set of keys $\mathcal{K} = \{k, k_{T,K}^{t \to k}\}$. The assumed condition Φ is the conjunction of the following formulae:

$$\Phi_0 : \forall X.\ \mathsf{New}(X, AKey) \supset \hat{X} = \hat{K}$$
$$\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))$$
$$\Phi_2 : \forall X, m.\ \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.m, k_0) \supset k_0 \in \{k, k^{t \to k}_{T,K}\}$$
$$\Phi_3 : \forall X, \hat{S_0}, \hat{C_0}.\ \mathsf{New}(X, AKey) \supset \neg\mathsf{SymEnc}(X, AKey.\hat{C_0}, k^{s \to t}_{S_0,X})$$
$$\Phi_4 : \forall X, t.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Sign}(X, M, sk_X) \wedge \mathsf{ContainsOpen}(M, AKey))$$

Observe that Φ is prefix closed. Now we present the formal proof of goodness:

Let, $[\mathbf{Client}_1]_{C'}$: $[\texttt{new}\ n'_1;\ \texttt{new}\ \tilde{n'}_1;$
$sigc' := \texttt{sign}\ t'_C.\tilde{n'}_1, sk_{C'};$
$\texttt{send}\ Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1;]_{C'}$

$$[\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n'_1) \wedge \mathsf{Send}(C', \hat{C}'.\hat{T}'.n'_1) \wedge \mathsf{Sign}(C', t'_C.\tilde{n'}_1, sk_{C'}) \quad (A.116)$$
$$\Phi_1, \Phi_4, (-1) \quad [\mathbf{Client}_1]_{C'}\ n'_1 \neq AKey \wedge \tilde{n'}_1 \neq AKey \quad (A.117)$$
$$\mathbf{G1}, \mathbf{G5}, \mathbf{G11}, (-1) \quad [\mathbf{Client}_1]_{C'}\ \mathsf{Good}(C', Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1, AKey, \mathcal{K}) \quad (A.118)$$
$$\mathbf{SG1\text{-}2}, (-1) \quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \quad (A.119)$$

Let, $[\mathbf{Client}_2]_{C'}$: $[\texttt{receive}\ encp'_{kc}.\hat{C}'.tgt'.enc'_{kc};$
$textp'_{kc} := \texttt{pkdec}\ encp'_{kc}, dk_{C'};$
$\texttt{match}\ textp'_{kc}\ \texttt{as}\ Cert_{K'}.sigk';$
$\texttt{verify}\ sigk', k'.ck', vk'_K;$
$\tilde{ck}' := \texttt{hash}\ Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1, k';$
$\texttt{match}\ \tilde{ck}'\ \texttt{as}\ ck';$
$text'_{kc} := \texttt{symdec}\ enc'_{kc}, k';$
$\texttt{match}\ text'_{kc}\ \texttt{as}\ AKey'.n'_1.t'_K.\hat{T}';]_{C'}$

$$\mathbf{SG1} \quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \quad (A.120)$$

Precondition θ_3 : $\mathsf{Good}(C', tgt', AKey, \mathcal{K})$

Let, $[\mathbf{Client}_3]_{C'}$: $[\texttt{new}\ n_2';\ enc_{ct}' := \texttt{symenc}\ \hat{C}', AKey';$
$\texttt{send}\ tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2';]_{C'}$

$$[\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n_2') \wedge \mathsf{Send}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2') \tag{A.121}$$

$\Phi_1, (-1)$ $$[\mathbf{Client}_3]_{C'}\ n_2' \neq AKey \tag{A.122}$$

$\mathbf{G1}, (-1)$ $$\theta_3\ [\texttt{new}\ n_2';]_{C'}\ \mathsf{Good}(C', tgt', AKey, \mathcal{K}) \wedge \mathsf{Good}(C', n_2', AKey, \mathcal{K}) \tag{A.123}$$

$\mathbf{G*}, (-1)$ $$\theta_3\ [\mathbf{Client}_3]_{C'}\ \mathsf{Good}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2', AKey, \mathcal{K}) \tag{A.124}$$

$\mathbf{SG1\text{-}2}, (-1)$ $$\theta_3 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{A.125}$$

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SendGood}(C', AKey, \mathcal{K})\ [\texttt{receive}\ \hat{C}', st', enc_{tc}';$
$text_{tc}' := \texttt{symdec}\ enc_{tc}', AKey'; \texttt{match}\ text_{tc}'\ \texttt{as}\ SKey'.n_2'.\hat{S}';]_{C'}$
$$\mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{A.126}$$

Precondition θ_5 : $\mathsf{Good}(C', st', AKey, \mathcal{K})$

$\cdots$ proof for following BS similar to (13) $\cdots$

$\theta_5 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})$ [
$enc_{cs}' := \texttt{symenc}\ \hat{C}'.t', SKey';$
$\texttt{send}\ st', enc_{cs}';]_{C'}$
$$\mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{A.127}$$

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SendGood}(C', AKey, \mathcal{K})\ [\texttt{receive}\ enc_{sc}';$
$text_{sc}' := \texttt{symdec}\ enc_{sc}', SKey'; \texttt{match}\ text_{sc}'\ \texttt{as}\ t';]_{C'}$
$$\mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{A.128}$$

Let, $[\mathbf{KAS}]_{K'}$: $[$`receive` $Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n_1'$;
`verify` $sigc', t_C'.\tilde{n_1'}, vk_{C'}$;
`new` k';`new` $AKey'$;
$ck' :=$ `hash` $Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n_1', k'$;
$sigk' :=$ `sign` $k'.ck', sk_{K'}$;
$encp_{kc}' :=$ `pkenc` $Cert_{K'}.sigk', pk_{C'}$;
$tgt' :=$ `symenc` $AKey'.\hat{C}', k_{T',K'}^{t\rightarrow k}$;
$enc_{kc}' :=$ `symenc` $AKey'.n_1'.t_{K'}.\hat{T}', k'$;
`send` $encp_{kc}'.\hat{C}'.tgt'.enc_{kc}';]_{K'}$

$[\mathbf{KAS}]_{K'}$ $\mathsf{Sign}(K', k'.ck', sk_{K'})$ (A.129)

$\Phi_4, (-1)$ $[\mathbf{KAS}]_{K'}$ $k' \neq AKey$ (A.130)

Case 1 : $AKey' = AKey$

$[\mathbf{KAS}]_{K'}$ $\mathsf{New}(K', AKey) \wedge \mathsf{SymEnc}(K', AKey.\hat{C}', k_{T',K'}^{t\rightarrow k})$
$\wedge\, \mathsf{SymEnc}(K', AKey.n_1'.t_{K'}.\hat{T}', k')$ (A.131)

$\Phi_2, (-1)$ $[\mathbf{KAS}]_{K'}$ $k' \in \{k, k_{T,K}^{t\rightarrow k}\} \wedge k_{T',K'}^{t\rightarrow k} \in \{k, k_{T,K}^{t\rightarrow k}\}$ (A.132)

(-1) $[\mathbf{KAS}]_{K'}$ $k' \in \mathcal{K} \wedge k_{T',K'}^{t\rightarrow k} \in \mathcal{K}$ (A.133)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}$ $\mathsf{Good}(K', encp_{kc}'.\hat{C}'.tgt'.enc_{kc}', AKey, \mathcal{K})$ (A.134)

Case 2 : $AKey' \neq AKey$

G2 $[$`receive` $Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n_1';]_{K'}$ $\mathsf{Good}(K', Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n_1', AKey, \mathcal{K})$ (A.135)

G6, (-1) $[$`receive` $Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n_1';]_{K'}$ $\mathsf{Good}(K', n_1', AKey, \mathcal{K})$ (A.136)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}$ $\mathsf{Good}(encp_{kc}'.\hat{C}'.tgt'.enc_{kc}', AKey, \mathcal{K})$ (A.137)

SG1-2, $(-4, -1)$ $\mathsf{SendGood}(K', AKey, \mathcal{K})$ $[\mathbf{KAS}]_{K'}$ $\mathsf{SendGood}(K', AKey, \mathcal{K})$ (A.138)

$$
\begin{aligned}
\text{Let, } [\mathbf{TGS}]_{T'} : & [\texttt{receive } enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;\\
& text'_{ct1} := \texttt{symdec } enc'_{ct1}, k^{t\to k}_{T',K'};\\
& \texttt{match } text'_{ct1} \texttt{ as } AKey'.\hat{C}';\\
& text'_{ct2} := \texttt{symdec } enc'_{ct2}, AKey';\\
& \texttt{match } text'_{ct2} \texttt{ as } \hat{C}';\\
& \texttt{new } SKey';\\
& st' := \texttt{symenc } SKey'.\hat{C}', k^{s\to t}_{S',T'};\\
& enc'_{tc} := \texttt{symenc } SKey'.n'_2.\hat{S}', AKey';\\
& \texttt{send } \hat{C}'.st'.enc'_{tc};]_{T'}
\end{aligned}
$$

$$\mathbf{G2}, \mathbf{G6} \quad [\texttt{receive } enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;]_{T'} \ \mathsf{Good}(T', n'_2, AKey, \mathcal{K}) \tag{A.139}$$

$$[\mathbf{TGS}]_{T'} \ \mathsf{New}(T', SKey') \wedge \mathsf{SymEnc}(T', SKey'.\hat{C}', k^{s\to t}_{S',T'}) \tag{A.140}$$

$$\Phi_3, (-1) \quad [\mathbf{TGS}]_{T'} \ SKey' \neq AKey \tag{A.141}$$

$$\mathbf{G1}, (-1) \quad [\cdots; \texttt{new } SKey';]_{T'} \ \mathsf{Good}(T', SKey', AKey, \mathcal{K}) \tag{A.142}$$

$$\mathbf{G}{*}, (-4, -1) \quad [\mathbf{TGS}]_{T'} \ \mathsf{Good}(T', \hat{C}'.st'.enc'_{tc}, AKey, \mathcal{K}) \tag{A.143}$$

$$\mathbf{SG1\text{-}2}, (-1) \quad \mathsf{SendGood}(T', AKey, \mathcal{K}) \ [\mathbf{TGS}]_{T'} \ \mathsf{SendGood}(T', AKey, \mathcal{K}) \tag{A.144}$$

Let, $[\mathbf{Server}]_{S'}$: $[\texttt{receive } enc'_{cs1}.enc'_{cs2};$
$text'_{cs1} := \texttt{symdec } enc'_{cs1}, k^{s\rightarrow t}_{S',T'}; \texttt{match } text'_{cs1} \texttt{ as } SKey'.\hat{C}';$
$text'_{cs2} := \texttt{symdec } enc'_{cs2}, SKey'; \texttt{match } enc'_{cs2} \texttt{ as } \hat{C}'.t';$
$enc'_{sc} := \texttt{symenc } t', SKey';$
$\texttt{send } enc'_{sc};]_{S'}$

Case 1:	$SKey' \in \mathcal{K}$	
$\mathbf{G7}, (-1)$	$[\cdots; enc'_{sc} := \texttt{symenc } t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$	(A.145)
Case 2:	$SKey' \notin \mathcal{K}$	(A.146)
$\mathbf{G2}, \mathbf{G6}$	$[\texttt{receive } enc'_{cs1}.enc'_{cs2};]_{S'}\ \mathsf{Good}(S', enc'_{cs2}, AKey, \mathcal{K})$	(A.147)
$\mathbf{G6}, \mathbf{G8}, (-1)$	$[\cdots; text'_{cs2} := \texttt{symdec } enc'_{cs2}, SKey'; \texttt{match } enc'_{cs2} \texttt{ as } \hat{C}'.t';]_{S'}$ $\mathsf{Good}(S', t', AKey, \mathcal{K})$	(A.148)
$\mathbf{G7}, (-1)$	$[\cdots; enc'_{sc} := \texttt{symenc } t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$	(A.149)
$(-4, -1)$	$[\mathbf{Server}]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$	(A.150)
$\mathbf{SG1\text{-}2}, (-1)$	$\mathsf{SendGood}(S', AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SendGood}(S', AKey, \mathcal{K})$	(A.151)
Theorem 12	$\Phi \supset \mathsf{Secretive}(AKey, \mathcal{K})$	(A.152)

We can derive from $AUTH^{client}_{kas}$, the actions in $[\mathbf{KAS}]_K$ and $AUTH^{client}_{tgs}$ that:

$$PKINIT \vdash [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$
$$PKINIT \vdash [\mathbf{KAS}]_K\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$
$$PKINIT \vdash [\mathbf{TGS}]_T\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$

The only principals having access to a key in $\mathcal{K}$ are $\hat{C}, \hat{K}$ and $\hat{T}$. All keys in $\mathcal{K}$ are level-0 as they are used only as keys. In addition, Φ_0 assumes that some thread of K generated $AKey$. Therefore, we have:

$$\mathsf{InInitSet}(X, AKey, \mathcal{K}) \equiv \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}$$
$$\mathsf{GoodInit}(AKey, \mathcal{K}) \equiv \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T})$$
$$\mathsf{GoodKeyFor}(AKey, \mathcal{K}) \equiv \mathsf{GoodKeyAgainst}(AKey, X) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}$$

Therefore, by axiom **GK**, we have:

$$\mathsf{Secretive}(AKey, \mathcal{K}) \wedge \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \Rightarrow \mathsf{GoodKeyAgainst}(X, AKey) \vee \hat{X} \in \{\hat{C}, \hat{K}, \hat{T}\}$$

Combining everything we have:

$$PKINIT \vdash SEC_{akey}^{client}, SEC_{akey}^{kas}, SEC_{akey}^{tgs}$$

A.2.7 Proofs of $SEC_{skey}^{client}, SEC_{skey}^{tgs}$

The proof follows the same structure as in section A.2.6 and its details are skipped. The assumed condition Φ is the conjunction of the following formulae:

$$\Phi_0 : \forall X.\ \mathsf{New}(X, SKey) \supset \hat{X} = \hat{T}$$

$$\Phi_1 : \forall X, M.\ \mathsf{New}(X, SKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, SKey))$$

$$\Phi_2 : \forall X, m.\ \mathsf{New}(X, SKey) \wedge \mathsf{SymEnc}(X, SKey.m, k_0) \supset k_0 \in \{AKey, k_{S,T}^{s \to t}\}$$

$$\Phi_3 : \forall X, \hat{T_0}, \hat{C_0}.\ \mathsf{New}(X, SKey) \supset \neg\mathsf{SymEnc}(X, SKey.\hat{C_0}, k_{T_0,X}^{t \to k})$$

$$\Phi_4 : \forall X, t.\ \mathsf{New}(X, SKey) \supset \neg(\mathsf{Sign}(X, M, sk_X) \wedge \mathsf{ContainsOpen}(M, SKey))$$

A.2.8 Proof of $AUTH_{tgs}^{client}$

We have derived in Appendix A.2.6 that on execution of the **Client** role by C, $\mathsf{GoodProtocol}(AKey, \mathcal{K}) \wedge \mathsf{GoodInit}(AKey, \mathcal{K})$ holds where $\mathcal{K} = \{k_{C,K}^{c \to k}, k_{T,K}^{t \to k}\}$. Therefore, by axiom **CTXL** we can derive that:

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(Z, E_{sym}[AKey](m), AKey) \Rightarrow \exists X.\ \mathsf{SymEnc}(X, m, AKey) \qquad \text{(A.153)}$$

Now, the thread $[\textbf{Client}]_C$ decrypts the message $E_{sym}[AKey](SKey.n_2.\hat{S})$. Hence we proceed:

$$
\begin{array}{rll}
 & [\textbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[AKey](SKey.n_2.\hat{S}), AKey) & \text{(A.154)}\\
(-1,1) & [\textbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, SKey.n_2.\hat{S}, AKey) & \text{(A.155)}\\
\text{Inst } X \mapsto X_0, (-1) & [\textbf{Client}]_C\ \mathsf{SymEnc}(X_0, SKey.n_2.\hat{S}, AKey) & \text{(A.156)}\\
(-1) & [\textbf{Client}]_C\ \mathsf{Possess}(X_0, AKey) & \text{(A.157)}\\
SEC^{client}_{AKey}, (-1) & \hat{X_0} = \hat{C} \wedge \hat{X_0} = \hat{K} \wedge \hat{X_0} = \hat{T} & \text{(A.158)}\\
\text{Inst }, AUTH^{client}_{kas} & [\textbf{Client}]_C\ \mathsf{New}(K, AKey) \wedge \forall m.\ \neg\mathsf{SymEnc}(K, m, AKey) & \text{(A.159)}\\
(-4,-1) & [\textbf{Client}]_C\ \neg\mathsf{New}(X_0, AKey) & \text{(A.160)}
\end{array}
$$

$$
\begin{array}{rll}
\textbf{HON} & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key'.n.\hat{S_0}, Key) \wedge \neg\mathsf{New}(X, Key) & \\
 & \supset \exists \hat{K_0}, \hat{C_0}.\ \mathsf{SymDec}(X, E_{sym}[k^{t\to k}_{X,K_0}](Key.\hat{C_0}), k^{t\to k}_{X,K_0}) & \\
 & \wedge\ \mathsf{Send}(X, \hat{C_0}.E_{sym}[k^{s\to t}_{S_0,X}](Key'.\hat{C_0}).E_{sym}[Key](Key'.n.\hat{S_0})) & \text{(A.161)}
\end{array}
$$

$$
\begin{array}{rll}
\text{Inst}, (-4,-1) & [\textbf{Client}]_C\ \mathsf{SymDec}(X_0, E_{sym}[k^{t\to k}_{X_0,K_0}](AKey.\hat{C_0}), k^{t\to k}_{X_0,K_0}) & \\
 & \wedge\ \mathsf{Send}(X_0, \hat{C_0}.E_{sym}[k^{s\to t}_{S,X_0}](SKey.\hat{C_0}).E_{sym}[AKey](SKey.n_2.\hat{S})) & \text{(A.162)}\\
\textbf{CTX0}, (-1) & [\textbf{Client}]_C\ \exists Y.\ \mathsf{SymEnc}(Y, AKey.\hat{C_0}, k^{t\to k}_{X_0,K_0}) & \text{(A.163)}\\
\text{Inst } Y \mapsto Y_0, (-1) & [\textbf{Client}]_C\ \mathsf{SymEnc}(Y_0, AKey.\hat{C_0}, k^{t\to k}_{X_0,K_0}) & \text{(A.164)}
\end{array}
$$

$$
\begin{array}{rll}
\textbf{HON} & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{W}, k^{t\to k}_{X,Z}) \supset \mathsf{New}(X, Key) & \text{(A.165)}
\end{array}
$$

$$
\begin{array}{rll}
(-4,-1) & [\textbf{Client}]_C\ \mathsf{New}(Y_0, AKey) & \text{(A.166)}\\
AUTH^{client}_{kas} & \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.\hat{W}, k^{t\to k}_{Y,Z}) & \\
 & \supset \hat{Y} = \hat{T} \wedge \hat{Z} = \hat{K} \wedge \hat{W} = \hat{C} & \text{(A.167)}\\
(10,-2,-1) & \hat{X_0} = \hat{T} \wedge \hat{K_0} = \hat{K} \wedge \hat{C_0} = \hat{C} & \text{(A.168)}\\
(A.162,-1) & [\textbf{Client}]_C\ \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s\to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S})) & \text{(A.169)}
\end{array}
$$

Appendix B

Diffie-Hellman-based Protocols

B.1 Proofs of DHINIT Security Properties

B.1.1 Proofs of $AUTH_{kas}^{client}, AUTH_{tgs}^{client}$

We describe below the key steps in the formal proof, focusing especially on the cryptographic assumptions on which they rely.

Part of the authentication after the first stage is achieved by verifying the signature of the KAS. This reasoning step depends on the CMA-security of the signature scheme which is formalized in the **SIG** axiom (see [36]). The client verifies that the KAS signed its Diffie-Hellman public value (gy) and the client's nonce ($\tilde{n_1}$). However, since the signature does not bind the identities of the client and the intended TGS, it only proves that the KAS observed the client's nonce and produced the DH exponential gy by exponentiating some nonce y. Formally, this fact is a program invariant and is established using an inductive invariant rule—the honesty rule **HON**.

The next step is to prove that the Diffie-Hellman key k can be used as an encryption key only by C and K. This is achieved by showing that C and K only send out "safe" messages containing the private exponents x and y. The precise definition of "safe" messages and the corresponding axioms are in Section 4.1. Now, we use the ciphertext integrity property of the encryption scheme, formally captured by the axiom **CTXG**. The soundness of this axiom, in addition to ciphertext integrity of the encryption scheme, requires the Computational DH assumption since k is produced by a Diffie-Hellman exchange. The soundness of **CTXG** is proved by modelling the key generation function as a random oracle. We now infer from the fact that the client decrypted the ciphertext $E_{sym}[k](AKey.n_1.\hat{T})$ and that the client did not produce it itself that it must have been produced by the KAS. We use the honesty rule to infer that a single thread only does a single encryption of this form and conclude that the same thread of KAS produced this message by encryption. At this point, we are assured that the KAS agrees on $\hat{T}, gx, n$ and $AKey$. However, it still doesn't agree on the identity of the client. As we saw from the counterexample in section 4.2.2, this is the best result possible after the first stage.

It turns out, as we will see in Theorem 22, that this partial authentication is sufficient to prove the secrecy of the authentication key ($AKey$) from the client's perspective. The proof requires the DDH assumption. Now, stronger authentication properties are proved from the second stage of the protocol once the client decrypts the message $E_{sym}[AKey]$ $(SKey.n_2.\hat{S})$. We infer that some thread of $\hat{C}, \hat{K}$ or $\hat{T}$ must have produced the encryption because of ciphertext integrity. Using an invariant to reason about the special form of this ciphertext, we conclude that the encrypting thread must have received a tgt containing $AKey$ and meant for itself. Since we have proved the secrecy of $AKey$ already, under the keys k and $k^{t\rightarrow k}_{T,K}$, we infer that this tgt must be keyed with one of k and $k^{t\rightarrow k}_{T,K}$ the holders of which—$\hat{C}$, $\hat{T}$ and $\hat{K}$—are honest. This reasoning is formally captured in the axiom **SDEC** whose soundness relies on the encryption scheme being IND-CCA secure. Now we use the honesty rule to infer that if an honest thread encrypted this message then it must have generated $AKey$; we know that thread is K. At this point, we conclude that the TGS agrees on the identity of the KAS. It is only left to prove that the TGS agrees on the identity of the client.

We use the honesty rule to infer that the TGS decrypted a message $E_{sym}[AKey](\hat{C}_0)$. Due to the secrecy of $AKey$, this encryption could only have been done by some thread of $\hat{C}$, $\hat{T}$ or $\hat{K}$. We again use the honesty rule to infer that some thread of $\hat{C}_0$ must have encrypted the message and it must have got $AKey$ by decrypting some other message with a Diffie-Hellman shared key. Using axiom

SDEC again and noting that k is the only DH key protecting $AKey$, we infer that this key must be k - so this narrows down the possible threads who could have done this encryption to only C, the generator of x and K, the generator of y. We again use the honesty rule to infer that K did not do this, hence C_0 must be C. Therefore the TGS agrees on the identity of the client. Since this also means the KAS sent the TGS a tgt containing the identity of $\hat{C}$, we also conclude that the KAS agrees on the identity of the client.

Reference to equations by negative numbers is relative to the current equation - e.g. (-1) refers to the last equation. Reference by positive numbers indicates the actual number of the equation. The "Inst" steps below signify instantiation of existential variables by constants - we expect it to be clear from the context which variables are instantiated.

$[\mathbf{Client}]_C\ \mathsf{Verify}(C, SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}), vk_K)$ (B.1)

$\mathbf{SIG}, (-1)$ $[\mathbf{Client}]_C\ \exists\eta.\ \mathsf{Sign}((\hat{K},\eta), \text{“}DHKey\text{”}.gy.\tilde{n_1}, sk_K)$ (B.2)

Inst $(\hat{K},\eta)\mapsto K$ $[\mathbf{Client}]_C\ \mathsf{Sign}(K, \text{“}DHKey\text{”}.gy.\tilde{n_1}, sk_K)$ (B.3)

$\mathbf{HON}, \mathbf{DH}*,$ $\mathbf{SDH}*$
$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Sign}(X, \text{“}DHKey\text{”}.gy_0.n, sk_X) \supset$
$\exists \hat{C_0}, \hat{T_0}, gx_0, m_0.$
$\mathsf{Receive}(X, Cert_{C_0}.SIG[C_0](\text{“}Auth\text{”}.HASH(\hat{C_0}.\hat{T_0}.m_0).n.gx_0).\hat{C_0}.\hat{T_0}.m_0) \wedge$
$\exists y_0, k_0.\ \mathsf{New}(X, y_0) \wedge gy_0 = g^{y_0} \wedge \mathsf{KeyGen}(X, k_0, y_0, gx_0) \wedge$
$\mathsf{Send}(X, Cert_X.sigk.E_{sym}[k^{t\to k}_{T_0,X}](AKey_0.\hat{C_0}).E_{sym}[k_0](AKey_0.m_0.\hat{T_0})) \wedge$
$\mathsf{SendDHGood}(X, y_0)$ (B.4)

$(-1,-2)$, Inst $[\mathbf{Client}]_C\ \mathsf{Receive}(K, Cert_{C_0}.SIG[C_0](\text{“}Auth\text{”}.HASH(\hat{C_0}.\hat{T_0}.m_0).\tilde{n_1}.gx_0).\hat{C_0}.\hat{T_0}.m_0) \wedge$
$\mathsf{New}(K, y) \wedge gy = g^y \wedge \mathsf{KeyGen}(K, k_0, y, gx_0) \wedge$
$\mathsf{Send}(K, Cert_K.sigk.E_{sym}[k^{t\to k}_{T_0,K}](AKey_0.\hat{C_0}).E_{sym}[k_0](AKey_0.m_0.\hat{T_0})) \wedge$
$\mathsf{SendDHGood}(K, y)$ (B.5)

$\mathbf{DH}*, \mathbf{SDH}*$ $[\mathbf{Client}]_C\ \mathsf{New}(C, x) \wedge \mathsf{SendDHGood}(C, x) \wedge \mathsf{KeyGen}(C, k, x, g^y)$ (B.6)

$(B.5, -1)$ $[\mathbf{Client}]_C\ \mathsf{DHSecretive}(C, K, k)$ (B.7)

$[\mathbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[k](AKey.n_1.\hat{T}), k)$ (B.8)

$\mathbf{CTXG}, (-2,-1)$ $[\mathbf{Client}]_C\ \exists X \in \{C, K\}.\ \mathsf{SymEnc}(X, AKey.n_1.\hat{T}, k)$ (B.9)

$[\mathbf{Client}]_C\ \neg\mathsf{SymEnc}(C, AKey.n_1.\hat{T}, k)$ (B.10)

$(-2,-1)$ $[\mathbf{Client}]_C\ \mathsf{SymEnc}(K, AKey.n_1.\hat{T}, k)$ (B.11)

$\mathbf{HON}$ $\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key_0.m_0.\hat{T_0}, k_0) \wedge \mathsf{SymEnc}(X, Key_1.m_1.\hat{T_1}, k_1) \supset$
$Key_0 = Key_1 \wedge m_0 = m_1 \wedge \hat{T_0} = \hat{T_1} \wedge k_0 = k_1$ (B.12)

$(B.5, -1)$ $[\mathbf{Client}]_C\ \mathsf{Receive}(K, Cert_{C_0}.SIG[C_0](\text{“}Auth\text{”}.HASH(\hat{C_0}.\hat{T}.n_1).\tilde{n_1}.gx).\hat{C_0}.\hat{T}.n_1) \wedge$
$\mathsf{New}(K, y) \wedge gy = g^y \wedge \mathsf{KeyGen}(K, k, y, gx) \wedge$
$\mathsf{Send}(K, Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}).E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C_0}).E_{sym}[k](AKey.n_1.\hat{T}))$
(B.13)

Let us call equation (B.13) $A\widetilde{UT}H_{kas}^{client}$. At this point, observing that the Client and KAS agree on each other's DH exponentials and the protocol property (proved as an invariant) that each party uses only one DH exponential from the peer to generate a key, we can establish the stronger condition $\mathsf{DHStrongSecretive}(C, K, k)$. In section B.1.3 we establish the trace secrecy property of AKey: $\mathsf{Secretive}(AKey, \{k, k_{T,K}^{t\to k}\})$ using $A\widetilde{UT}H_{kas}^{client}$. Now we use this result to proceed as follows:

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[AKey](SKey.n_2.\hat{S}), AKey) \tag{B.14}$$

$$\mathbf{CTXL}, (-1,1) \qquad [\mathbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, SKey.n_2.\hat{S}, AKey) \tag{B.15}$$

$$\text{Inst } X \mapsto X_0, (-1) \qquad [\mathbf{Client}]_C\ \mathsf{SymEnc}(X_0, SKey.n_2.\hat{S}, AKey) \tag{B.16}$$

$$(-1) \qquad [\mathbf{Client}]_C\ \mathsf{Possess}(X_0, AKey) \tag{B.17}$$

$$SEC_{AKey}^{client}, (-1) \qquad \hat{X_0} = \hat{C} \wedge \hat{X_0} = \hat{K} \wedge \hat{X_0} = \hat{T} \tag{B.18}$$

$$A\widetilde{UT}H_{kas}^{client} \qquad [\mathbf{Client}]_C\ \mathsf{New}(K, AKey) \tag{B.19}$$

$$(-4,-1) \qquad [\mathbf{Client}]_C\ \neg\exists x_0, gy_0.\ \mathsf{KeyGen}(X_0, AKey, x_0, gy_0) \tag{B.20}$$

$$\begin{aligned}\mathbf{HON} \qquad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key'.n.\hat{S_0}, Key) \wedge \neg\exists x_0, gy_0.\ \mathsf{KeyGen}(X, Key, x_0, gy_0) \supset \\ & \exists \hat{K_0}, \hat{C_0}.\ \mathsf{SymDec}(X, E_{sym}[k_{X,K_0}^{t\to k}](Key.\hat{C_0}), k_{X,K_0}^{t\to k}) \wedge \\ & \mathsf{SymDec}(X, E_{sym}[Key](\hat{C_0}), Key) \wedge \\ & \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{S_0,X}^{s\to t}](Key'.\hat{C_0}).E_{sym}[Key](Key'.n.\hat{S_0}))\end{aligned} \tag{B.21}$$

$$\begin{aligned}\text{Inst}, (-4,-1) \qquad & [\mathbf{Client}]_C\ \mathsf{SymDec}(X_0, E_{sym}[k_{X_0,K_0}^{t\to k}](AKey.\hat{C_1}), k_{X_0,K_0}^{t\to k}) \wedge \\ & \mathsf{SymDec}(X_0, E_{sym}[AKey](\hat{C_1}), AKey) \wedge \\ & \mathsf{Send}(X_0, \hat{C_1}.E_{sym}[k_{S,X_0}^{s\to t}](SKey.\hat{C_1}).E_{sym}[AKey](SKey.n_2.\hat{S}))\end{aligned} \tag{B.22}$$

$$\mathbf{SDEC}, \mathbf{CTX0}, (-1) \qquad [\mathbf{Client}]_C\ \exists Y.\ \mathsf{SymEnc}(Y, AKey.\hat{C_1}, k_{X_0,K_0}^{t\to k}) \tag{B.23}$$

$$\text{Inst } Y \mapsto Y_0, (-1) \qquad [\mathbf{Client}]_C\ \mathsf{SymEnc}(Y_0, AKey.\hat{C_1}, k_{X_0,K_0}^{t\to k}) \tag{B.24}$$

$$\mathbf{HON} \qquad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{W}, k_{X,Z}^{t\to k}) \supset \mathsf{New}(X, Key) \tag{B.25}$$

$$(-4,-1) \qquad [\mathbf{Client}]_C\ \mathsf{New}(Y_0, AKey) \tag{B.26}$$

$$\begin{aligned}A\widetilde{UT}H_{kas}^{client} \qquad & \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.\hat{W}, k_{Y,Z}^{t\to k}) \\ & \supset \hat{Y} = \hat{T} \wedge \hat{Z} = \hat{K} \wedge \hat{W} = \hat{C_0}\end{aligned} \tag{B.27}$$

$$(10,-2,-1) \qquad \hat{X_0} = \hat{T} \wedge \hat{K_0} = \hat{K} \wedge \hat{C_1} = \hat{C_0} \tag{B.28}$$

$(B.22, -1)$	$[\textbf{Client}]_C\ \mathsf{SymDec}(T, E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C_0}), k^{t\to k}_{T,K}) \wedge$	
	$\mathsf{SymDec}(T, E_{sym}[AKey](\hat{C_0}), AKey) \wedge$	
	$\mathsf{Send}(T, \hat{C_0}.E_{sym}[k^{s\to t}_{S,T}](SKey.\hat{C_0}).E_{sym}[AKey](SKey.n_2.\hat{S}))$	(B.29)
$\mathbf{CTX}*, (B.29)$	$[\textbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, \hat{C_0}, AKey)$	(B.30)
$SEC^{client}_{AKey}, (-1)$	$[\textbf{Client}]_C\ \mathsf{SymEnc}(X_2, \hat{C_0}, AKey) \wedge (\hat{X_2} = \hat{C} \vee \hat{X_2} = \hat{K} \vee \hat{X_2} = \hat{T})$	(B.31)
$\mathbf{HON}$	$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, \hat{Y}, Key) \supset \hat{Y} = \hat{X} \wedge$	
	$\exists k_0.\ \mathsf{DHKeyGen}(X, k_0) \wedge \mathsf{SymDec}(X, E_{sym}[k_0](Key.n.\hat{T_0}), k_0)$	(B.32)
$(-2, -1)$	$[\textbf{Client}]_C\ \mathsf{SymEnc}(C_0, \hat{C_0}, AKey) \wedge (\hat{C_0} = \hat{C} \vee \hat{C_0} = \hat{K} \vee \hat{C_0} = \hat{T})$	(B.33)
SEC^{client}_{AKey}	$[\textbf{Client}]_C\ \mathsf{Secretive}(AKey, \{k, k^{t\to k}_{T,K}\}) \wedge \forall X.\neg\mathsf{DHKeyGen}(X, k^{t\to k}_{T,K}) \wedge$	
	$\forall X.\ \mathsf{DHKeyGen}(X, k) \supset (X = C \vee X = K)$	(B.34)
$\mathbf{SDEC}, (-1)$	$[\textbf{Client}]_C\ C_0 = C \vee C_0 = K$	(B.35)
	$[\textbf{Client}]_C\ \neg\mathsf{SymEnc}(K, \hat{K}, AKey)$	(B.36)
	$[\textbf{Client}]_C\ C_0 = C$	(B.37)
$(B.13, -1)$	$[\textbf{Client}]_C\ \exists\eta.\ \mathsf{Send}((\hat{K}, \eta), Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}).$	
	$E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C_0}).E_{sym}[k](AKey.n_1.\hat{T}))$	(B.38)
$(B.29, -2)$	$[\textbf{Client}]_C\ \exists\eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k^{s\to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))$	(B.39)

B.1.2 Proofs of $AUTH^{tgs}_{kas}$ and $AUTH^{server}_{tgs}$

We first give a template proof of $[\mathbf{Role}]_X\ \mathsf{Hon}(\hat{X}, \hat{Y}) \supset \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y})$, where **Role** decrypts the term $E_{sym}[k^{type}_{X,Y}](M)$. Reference to equations by negative numbers is relative to the current equation - e.g. (-1) refers to the last equation. Reference by positive numbers indicates the actual number of the equation.

$$\text{Hypothesis} \quad [\mathbf{Role}]_X\ \mathsf{SymDec}(X, E_{sym}[k^{type}_{X,Y}](M), k^{type}_{X,Y}) \tag{B.40}$$

$$\mathbf{CTX0}, (-2,-1) \quad [\mathbf{Role}]_X\ \exists Z.\ \mathsf{SymEnc}(Z, M, k^{type}_{X,Y}) \tag{B.41}$$

$$\text{Inst } Z \mapsto Z_0, (-1) \quad [\mathbf{Role}]_X\ \mathsf{SymEnc}(Z_0, M, k^{type}_{X,Y}) \tag{B.42}$$

$$\mathbf{A2}, (-1) \quad [\mathbf{Role}]_X\ \mathsf{Possess}(Z_0, k^{type}_{X,Y}) \tag{B.43}$$

$$\mathsf{Hon}(\hat{X}, \hat{Y}), \Gamma_0, (-1) \quad [\mathbf{Role}]_X\ \hat{Z}_0 = \hat{X} \vee \hat{Z}_0 = \hat{Y} \tag{B.44}$$

$$\begin{aligned}(-4,-1) \quad & [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{X}, \eta), M, k^{type}_{X,Y}) \\ & \vee\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y})\end{aligned} \tag{B.45}$$

Case 1 : $\hat{X} = \hat{Y}$

$$(-1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y}) \tag{B.46}$$

Case 2 : $\hat{X} \neq \hat{Y}$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}_0) \wedge \hat{X}_0 \neq \hat{Y}_0 \supset \neg\mathsf{SymEnc}(X_0, M_0, k^{type}_{X_0,Y_0}) \tag{B.47}$$

$$\mathsf{Hon}(\hat{X}), (-1) \quad [\mathbf{Role}]_X\ \neg\exists \eta.\ \mathsf{SymEnc}((\hat{X}, \eta), M, k^{type}_{X,Y}) \tag{B.48}$$

$$(-4,-1) \quad [\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y}) \tag{B.49}$$

Instantiating for $AUTH_{kas}^{tgs}$:

$$[\mathbf{TGS}]_T\ \exists \eta.\ \mathsf{SymEnc}((\hat{K},\eta), AKey.\hat{C}, k_{T,K}^{t\to k}) \tag{B.50}$$

$$\begin{aligned}
\mathbf{HON}\quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C}_0, k_{Y,X}^{t\to k}) \supset \\
& \quad \exists k_0, gy_0, \tilde{n}, n.\ \mathsf{Send}(X, Cert_X.SIG[sk_X](\text{“}DHKey\text{”}.gy_0.\tilde{n}). \\
& \quad E_{sym}[k_{Y,X}^{t\to k}](Key.\hat{C}_0).E_{sym}[k_0](Key.n.\hat{T}))
\end{aligned} \tag{B.51}$$

$$\begin{aligned}
\mathsf{Hon}(\hat{K}), (-2,-1)\quad & [\mathbf{TGS}]_T\ \exists \eta, k, gy, \tilde{n_1}, n_1.\ \mathsf{Send}((\hat{K},\eta), Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}). \\
& \quad E_{sym}[k_{T,K}^{t\to k}](AKey.\hat{C}).E_{sym}[k](AKey.n_1.\hat{T}))
\end{aligned} \tag{B.52}$$

$$(-1)\quad AUTH_{kas}^{tgs} \tag{B.53}$$

Instantiating for $AUTH_{tgs}^{server}$:

$$[\mathbf{Server}]_S\ \exists \eta.\ \mathsf{SymEnc}((\hat{T},\eta), SKey.\hat{C}, k_{S,T}^{s\to t}) \tag{B.54}$$

$$\begin{aligned}
\mathbf{HON}\quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C}_0, k_{Y,X}^{s\to t}) \\
& \quad \supset \exists n, Key'.\ \mathsf{Send}(X, \hat{C}_0.E_{sym}[k_{Y,X}^{s\to t}](Key.\hat{C}_0).E_{sym}[Key'](Key.n.\hat{Y}))
\end{aligned} \tag{B.55}$$

$$\begin{aligned}
\mathsf{Hon}(\hat{T}), (-2,-1)\quad & [\mathbf{Server}]_S\ \exists \eta, n, Key'.\ \mathsf{Send}((\hat{T},\eta), \hat{C}.E_{sym}[k_{S,T}^{s\to t}](SKey.\hat{C}). \\
& \quad E_{sym}[Key'](SKey.n.\hat{S}))
\end{aligned} \tag{B.56}$$

$$(-1)\quad AUTH_{tgs}^{server} \tag{B.57}$$

B.1.3 Proofs of $SEC_{akey}^{client}, SEC_{akey}^{kas}, SEC_{akey}^{tgs}$

The main idea is to prove by induction over the steps of the protocol that $AKey$ occurs on the network only as an encryption key or as a payload protected by encryption with the Diffie-Hellman key k or the pre-shared key $k_{T,K}^{t\to k}$. Formally, this step is carried out using the secrecy induction rule IND_{GOOD}. We therefore infer that $AKey$ is good for use as an encryption key using the axiom **GK**.

The assumed condition Φ is the conjunction of the following formulae:

$$\Phi_0 : \forall X.\ \mathsf{New}(X, AKey) \supset \hat{X} = \hat{K}$$
$$\Phi_1 : \forall X, M.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, AKey))$$
$$\Phi_2 : \forall X, m.\ \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.m, k_0) \supset k_0 \in \{k, k^{t\to k}_{T,K}\}$$
$$\Phi_3 : \forall X, \hat{S_0}, \hat{C_0}.\ \mathsf{New}(X, AKey) \supset \neg\mathsf{SymEnc}(X, AKey.\hat{C_0}, k^{s\to t}_{S_0,X})$$
$$\Phi_4 : \forall X.\ \mathsf{New}(X, AKey) \supset \neg(\mathsf{Sign}(X, M, sk_X) \wedge \mathsf{ContainsOpen}(M, AKey))$$
$$\Phi_5 : \forall X.\ \mathsf{New}(X, AKey) \supset \neg\mathsf{Expg}(X, AKey)$$

Observe that Φ is prefix closed. Recall that the predicate $\mathsf{ContainsOpen}(m, a)$ asserts that a can be obtained from m by a series of unpairings only.

Now we present the formal proof of goodness:

Let, $[\mathbf{Client}_1]_{C'}$: $[\texttt{new } n'_1; \texttt{ new } \tilde{n'}_1;$
$chksum' := \texttt{hash } \hat{C'}.\hat{T'}.n'_1;$
$sigc' := \texttt{sign } \text{“}Auth\text{”}.chksum'.\tilde{n'}_1.gx, sk_{C'};$
$\texttt{send } Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n'_1;]_{C'}$

$$[\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n'_1) \wedge \mathsf{Send}(C', \hat{C'}.\hat{T'}.n'_1) \wedge \mathsf{Sign}(C', \text{“}Auth\text{”}.chksum'.\tilde{n'}_1.gx, sk_{C'}) \quad \text{(B.58)}$$

$\Phi_1, \Phi_4, (-1)$ $\quad [\mathbf{Client}_1]_{C'}\ n'_1 \neq AKey \wedge \tilde{n'}_1 \neq AKey$ (B.59)

$\mathbf{G1}, \mathbf{G5}, \mathbf{G11}, (-1)$ $\quad [\mathbf{Client}_1]_{C'}\ \mathsf{Good}(C', Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n'_1, AKey, \mathcal{K})$ (B.60)

$\mathbf{SG1\text{-}2}, (-1)$ $\quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K})$ (B.61)

Let, $[\mathbf{Client}_2]_{C'}$: $[\texttt{receive } Cert_{K'}.sigk'.\hat{C'}.tgt'.enc'_{kc};$
$\texttt{verify } sigk', \text{“}DHKey\text{”}.gy'.\tilde{n'_1}, vk_{K'};$
$k' := \texttt{dhkeygen } gy', x';$
$text'_{kc} := \texttt{symdec } enc'_{kc}, k';$
$\texttt{match } text'_{kc} \texttt{ as } AKey'.n'_1.\hat{T'};]_{C'}$

$\mathbf{SG1}$ $\quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K})$ (B.62)

Precondition θ_3 : $\mathsf{Good}(C', tgt', AKey, \mathcal{K})$

Let, $[\mathbf{Client}_3]_{C'}$: $[\texttt{new}\ n_2';\ enc_{ct}' := \texttt{symenc}\ \hat{C}', AKey';$
$\texttt{send}\ tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2';]_{C'}$

$$[\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n_2') \wedge \mathsf{Send}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2') \tag{B.63}$$

$\Phi_1, (-1)$ $$[\mathbf{Client}_3]_{C'}\ n_2' \neq AKey \tag{B.64}$$

$\mathbf{G1}, (-1)$ $$\theta_3\ [\texttt{new}\ n_2';]_{C'}\ \mathsf{Good}(C', tgt', AKey, \mathcal{K}) \wedge \mathsf{Good}(C', n_2', AKey, \mathcal{K}) \tag{B.65}$$

$\mathbf{G*}, (-1)$ $$\theta_3\ [\mathbf{Client}_3]_{C'}\ \mathsf{Good}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2', AKey, \mathcal{K}) \tag{B.66}$$

$\mathbf{SG1\text{-}2}, (-1)$ $$\theta_3 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{B.67}$$

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SendGood}(C', AKey, \mathcal{K})\ [\texttt{receive}\ \hat{C}', st', enc_{tc}';$
$text_{tc}' := \texttt{symdec}\ enc_{tc}', AKey'; \texttt{match}\ text_{tc}'\ \texttt{as}\ SKey'.n_2'.\hat{S}';]_{C'}$
$$\mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{B.68}$$

Precondition θ_5 : $\mathsf{Good}(C', st', AKey, \mathcal{K})$

$\cdots$ proof for following BS similar to (13) $\cdots$

$\theta_5 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [$
$enc_{cs}' := \texttt{symenc}\ \hat{C}'.t', SKey';$
$\texttt{send}\ st', enc_{cs}';]_{C'}$
$$\mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{B.69}$$

$\cdots$ proof for following BS similar to (5) $\cdots$

$\mathsf{SendGood}(C', AKey, \mathcal{K})\ [\texttt{receive}\ enc_{sc}';$
$text_{sc}' := \texttt{symdec}\ enc_{sc}', SKey'; \texttt{match}\ text_{sc}'\ \texttt{as}\ t';]_{C'}$
$$\mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{B.70}$$

Let, $[\mathbf{KAS}]_{K'}$: $[\texttt{receive}\ Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n_1';$
$\texttt{verify}\ sigc', \text{“}Auth\text{”}.chksum'.\tilde{n_1'}.gx', vk_{C'};$
$chk' := \texttt{hash}\ \hat{C'}.\hat{T'}.n_1';$
$\texttt{new}\ AKey';$
$\texttt{new}\ y'; gy' := \texttt{expg}\ y'; k' := \texttt{dhkeygen}\ gx', y';$
$sigk' := \texttt{sign}\ \text{“}DHKey\text{”}.gy'.\tilde{n_1'}, sk_{K'};$
$tgt' := \texttt{symenc}\ AKey'.\hat{C'}, k^{t\to k}_{T',K'};$
$enc_{kc}' := \texttt{symenc}\ AKey'.n_1'.\hat{T'}, k';$
$\texttt{send}\ Cert_{K'}.sigk'.\hat{C'}.tgt'.enc_{kc}';]_{K'}$ (B.71)

$[\mathbf{KAS}]_{K'}\ \mathsf{New}(K', y') \wedge \mathsf{Expg}(K', y')$ (B.72)

$\Phi_5, (-1)$ $[\mathbf{KAS}]_{K'}\ y' \neq AKey$ (B.73)

(-1) $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(K', y', AKey, \mathcal{K})$ (B.74)

Case 1 : $AKey' = AKey$

$[\mathbf{KAS}]_{K'}\ \mathsf{New}(K', AKey) \wedge \mathsf{SymEnc}(K', AKey.\hat{C'}, k^{t\to k}_{T',K'})$
$\wedge\ \mathsf{SymEnc}(K', AKey.n_1'.\hat{T'}, k')$ (B.75)

$\Phi_2, (-1)$ $[\mathbf{KAS}]_{K'}\ k' \in \{k, k^{t\to k}_{T,K}\} \wedge k^{t\to k}_{T',K'} \in \{k, k^{t\to k}_{T,K}\}$ (B.76)

(-1) $[\mathbf{KAS}]_{K'}\ k' \in \mathcal{K} \wedge k^{t\to k}_{T',K'} \in \mathcal{K}$ (B.77)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(K', Cert_{K'}.sigk'.\hat{C'}.tgt'.enc_{kc}', AKey, \mathcal{K})$ (B.78)

Case 2 : $AKey' \neq AKey$

$\mathbf{G2}$ $[\texttt{receive}\ Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n_1';]_{K'}\ \mathsf{Good}(K', Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n_1', AKey, \mathcal{K})$ (B.79)

$\mathbf{G6}, (-1)$ $[\texttt{receive}\ Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n_1';]_{K'}\ \mathsf{Good}(K', n_1', AKey, \mathcal{K})$ (B.80)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(K', Cert_{K'}.sigk'.\hat{C'}.tgt'.enc_{kc}', AKey, \mathcal{K})$ (B.81)

$\mathbf{SG1\text{-}2}, (-4, -1)$ $\mathsf{SendGood}(K', AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SendGood}(K', AKey, \mathcal{K})$ (B.82)

Let, $[\mathbf{TGS}]_{T'}$: $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;$
$text'_{ct1} := \texttt{symdec}\ enc'_{ct1}, k^{t\to k}_{T',K'};$
$\texttt{match}\ text'_{ct1}\ \texttt{as}\ AKey'.\hat{C}';$
$text'_{ct2} := \texttt{symdec}\ enc'_{ct2}, AKey';$
$\texttt{match}\ text'_{ct2}\ \texttt{as}\ \hat{C}';$
$\texttt{new}\ SKey';$
$st' := \texttt{symenc}\ SKey'.\hat{C}', k^{s\to t}_{S',T'};$
$enc'_{tc} := \texttt{symenc}\ SKey'.n'_2.\hat{S}', AKey';$
$\texttt{send}\ \hat{C}'.st'.enc'_{tc};]_{T'}$

G2, G6 $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;]_{T'}\ \mathsf{Good}(T', n'_2, AKey, \mathcal{K})$ (B.83)

$[\mathbf{TGS}]_{T'}\ \mathsf{New}(T', SKey') \wedge \mathsf{SymEnc}(T', SKey'.\hat{C}', k^{s\to t}_{S',T'})$ (B.84)

$\Phi_3, (-1)$ $[\mathbf{TGS}]_{T'}\ SKey' \neq AKey$ (B.85)

G1, (-1) $[\cdots;\texttt{new}\ SKey';]_{T'}\ \mathsf{Good}(T', SKey', AKey, \mathcal{K})$ (B.86)

G$*$, $(-4, -1)$ $[\mathbf{TGS}]_{T'}\ \mathsf{Good}(T', \hat{C}'.st'.enc'_{tc}, AKey, \mathcal{K})$ (B.87)

SG1-2, (-1) $\mathsf{SendGood}(T', AKey, \mathcal{K})\ [\mathbf{TGS}]_{T'}\ \mathsf{SendGood}(T', AKey, \mathcal{K})$ (B.88)

Let, $[\mathbf{Server}]_{S'}$: $[\texttt{receive}\ enc'_{cs1}.enc'_{cs2};$
$text'_{cs1} := \texttt{symdec}\ enc'_{cs1}, k^{s\to t}_{S',T'};\texttt{match}\ text'_{cs1}\ \texttt{as}\ SKey'.\hat{C}';$
$text'_{cs2} := \texttt{symdec}\ enc'_{cs2}, SKey';\texttt{match}\ enc'_{cs2}\ \texttt{as}\ \hat{C}'.t';$
$enc'_{sc} := \texttt{symenc}\ t', SKey';$
$\texttt{send}\ enc'_{sc};]_{S'}$

Case 1: $SKey' \in \mathcal{K}$

G7, (-1) $[\cdots;enc'_{sc} := \texttt{symenc}\ t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (B.89)

Case 2: $SKey' \notin \mathcal{K}$ (B.90)

G2, G6 $[\texttt{receive}\ enc'_{cs1}.enc'_{cs2};]_{S'}\ \mathsf{Good}(S', enc'_{cs2}, AKey, \mathcal{K})$ (B.91)

G6, G8, (-1) $[\cdots;text'_{cs2} := \texttt{symdec}\ enc'_{cs2}, SKey';\texttt{match}\ enc'_{cs2}\ \texttt{as}\ \hat{C}'.t';]_{S'}$
$\mathsf{Good}(S', t', AKey, \mathcal{K})$ (B.92)

G7, (-1) $[\cdots;enc'_{sc} := \texttt{symenc}\ t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (B.93)

$(-4, -1)$ $[\mathbf{Server}]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (B.94)

SG1-2, (-1) $\mathsf{SendGood}(S', AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SendGood}(S', AKey, \mathcal{K})$ (B.95)

Theorem 12 $\Phi \supset \mathsf{Secretive}(AKey, \mathcal{K})$ (B.96)

We can derive from $\tilde{AUTH}^{client}_{kas}$, the actions in $[\mathbf{KAS}]_K$ and $AUTH^{tgs}_{kas}$ that:

$$\begin{aligned}
&DHINIT \vdash [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset \Phi\\
&DHINIT \vdash [\mathbf{KAS}]_K\ \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset \Phi\\
&DHINIT \vdash [\mathbf{TGS}]_T\ \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset \Phi
\end{aligned}$$

The only principals having access to a key in $\mathcal{K}$ are $\hat{C},\hat{K}$ and $\hat{T}$. All keys in $\mathcal{K}$ are level-0 as they are used only as keys. In addition, Φ_0 assumes that some thread of K generated $AKey$. Therefore, we have:

$$\begin{aligned}
\mathsf{InInitSet}(X, AKey, \mathcal{K}) &\equiv \hat{X} \in \{\hat{C},\hat{K},\hat{T}\}\\
\mathsf{GoodInit}(AKey, \mathcal{K}) &\equiv \mathsf{Hon}(\hat{C},\hat{K},\hat{T})\\
\mathsf{GoodKeyFor}(AKey, \mathcal{K}) &\equiv \mathsf{GoodKeyAgainst}(AKey, X) \vee \hat{X} \in \{\hat{C},\hat{K},\hat{T}\}
\end{aligned}$$

Therefore, by axiom **GK**, we have:

$$\mathsf{Secretive}(AKey, \mathcal{K}) \wedge \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \Rightarrow \mathsf{GoodKeyAgainst}(X, AKey) \vee \hat{X} \in \{\hat{C},\hat{K},\hat{T}\}$$

Combining everything we have:

$$DHINIT \vdash SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$$

B.1.4 Proofs of $SEC^{client}_{skey}, SEC^{tgs}_{skey}$

The proof follows the same structure as in section B.1.3 and its details are skipped. The assumed condition Φ is the conjunction of the following formulae:

$$\begin{aligned}
&\Phi_0 : \forall X.\ \mathsf{New}(X, SKey) \supset \hat{X} = \hat{T}\\
&\Phi_1 : \forall X, M.\ \mathsf{New}(X, SKey) \supset \neg(\mathsf{Send}(X, M) \wedge \mathsf{ContainsOpen}(M, SKey))\\
&\Phi_2 : \forall X, m.\ \mathsf{New}(X, SKey) \wedge \mathsf{SymEnc}(X, SKey.m, k_0) \supset k_0 \in \{AKey, k^{s\to t}_{S,T}\}\\
&\Phi_3 : \forall X, \hat{T}_0, \hat{C}_0.\ \mathsf{New}(X, SKey) \supset \neg\mathsf{SymEnc}(X, SKey.\hat{C}_0, k^{t\to k}_{T_0,X})\\
&\Phi_4 : \forall X.\ \mathsf{New}(X, SKey) \supset \neg(\mathsf{Sign}(X, M, sk_X) \wedge \mathsf{ContainsOpen}(M, SKey))\\
&\Phi_5 : \forall X.\ \mathsf{New}(X, SKey) \supset \neg\mathsf{Expg}(X, SKey)
\end{aligned}$$

Bibliography

[1] IEEE P802.11i/D10.0. Medium Access Control (MAC) security enhancements, amendment 6 to IEEE Standard for local and metropolitan area networks part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications., April 2004.

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

[3] Pedro Adão, Gergei Bana, and Andre Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *Proc. of the 18th IEEE Computer Security Foudnations Workshop*, pages 170–184, 2005.

[4] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Just Fast Keying: Key agreement in a hostile internet. *ACM Trans. Inf. Syst. Security*, 7(4):1–30, 2004.

[5] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. IEEE Symposium on Security and Privacy*, pages 171–182. IEEE, 2005.

[6] Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Cryptographically sound security proofs for basic and public-key kerberos. In *Proceedings of 11th European Symposium on Research in Computer Security*, 2006. To appear.

[7] Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, and Mathieu Turuani. Compositional analysis of contract signing protocols. In *Proceedings of 18th IEEE Computer Security Foundations Workshop*. IEEE, 2005. To appear.

[8] Michael Backes and Birgit Pfitzmann. Limits of the cryptographic realization of XOR. In *Proc. of the 10th European Symposium on Research in Computer Security*. Springer-Verlag, 2005.

[9] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.

[10] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the reactive simulatability/UC of Dolev-Yao models with hashes. In *Proc. of the 11th European Symposium on Research in Computer Security*. Springer-Verlag, 2006.

[11] Mathieu Baudet, Véronique Cortier, and Steve Kremer. Computationally Sound Implementations of Equational Theories against Passive Adversaries. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663, Lisboa, Portugal, July 2005. Springer.

[12] Giampaolo Bella and Lawrence C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, editor, *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 361–375, Louvain-la-Neuve, Belgium, September 1998. Springer-Verlag LNCS 1485.

[13] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.

[14] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, pages 531–545, 2000.

[15] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[16] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93)*, pages 232–249. Springer-Verlag, 1994.

[17] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography. Lecture Notes, 2001.

[18] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, 2006.

[19] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *CRYPTO*, pages 537–554, 2006.

[20] A. Boldyreva and V. Kumar. Provable-security analysis of authenticated encryption in Kerberos. In *Proc. IEEE Security and Privacy*, 2007.

[21] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazare, and Bogdan Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In *Advances in Cryptology - CRYPTO 2007, Proceedings*, 2007.

[22] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. Verifying confidentiality and authentication in kerberos 5. In *ISSS*, pages 1–24, 2003.

[23] Ed. C. Kaufman. Internet Key Exchange (IKEv2) Protocol, 2005. RFC.

[24] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proc. of EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474, 2001.

[25] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Proceedings*, pages 19–40, 2001.

[26] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *Theory of Cryptography Conference - Proceedings of TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer-Verlag, 2006.

[27] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key kerberos. Technical report.

[28] Iliano Cervesato, Catherine Meadows, and Dusko Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *CSFW*, pages 48–61, 2005.

[29] Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. Key derivation and randomness extraction. Cryptology ePrint Archive, Report 2005/061, 2005. `http://eprint.iacr.org/`.

[30] Veronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 157–171. Springer-Verlag, 2005.

[31] Anupam Datta, Ante Derek, John C. Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the impossibility of realizable ideal functionality. In *Theory of Cryptography Conference - Proceedings of TCC 2006*, pages 360–379, 2006.

[32] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[33] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.

[34] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol Composition Logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[35] Anupam Datta, Ante Derek, John C. Mitchell, Vitaly Shmatikov, and Mathieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05)*, Lecture Notes in Computer Science. Springer-Verlag, pages 16–29, 2005.

[36] Anupam Datta, Ante Derek, John C. Mitchell, and Bogdan Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proceedings of 19th IEEE Computer Security Foundations Workshop*, pages 321–334. IEEE, 2006.

[37] Anupam Datta, Joseph Halpern, John C. Mitchell, Riccardo Pucella, and Arnab Roy. Reasoning about Conditional Probability and Concrete Security in Protocol Proofs (Work in Progress). In *Workshop on Formal and Computational Cryptography (FCC)*, Pittsburgh, 2008.

[38] Rob Delicata and Steve A. Schneider. Towards the rank function verification of protocols that use temporary secrets. In *Proceedings of the Workshop on Issues in the Theory of Security: WITS '04*, 2004.

[39] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[40] Nancy Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:677–721, 2003.

[41] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.

[42] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. IETF Internet draft, November 18 1996.

[43] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[44] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.

[45] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, 2005. to appear.

[46] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of ieee 802.11i and tls. In *ACM Conference on Computer and Communications Security*, pages 2–15, 2005.

[47] James Heather. Strand spaces and rank functions: More than distant cousins. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, page 104, 2002.

[48] Jonathan Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 234–247. IEEE, 2003.

[49] Jonathan Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, MIT, 2004.

[50] Romain Janvier, Laurent Mazare, and Yassine Lakhnech. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 172–185. Springer-Verlag, 2005.

[51] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *FSE*, pages 284–299, 2000.

[52] J. Kohl and B. Neuman. The kerberos network authentication service, 1991. RFC.

[53] Y. Lakhnech and L. Mazaré. Computationally sound verifiation of security protocols using Diffie-Hellman exponentiation. Cryptology ePrint Archive: Report 2005/097, 2005.

[54] Patrick D. Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. Probabilistic polynomial-time equivalence and security protocols. In *Formal Methods World Congress, vol. I*, number 1708 in Lecture Notes in Computer Science, pages 776–793. Springer-Verlag, 1999.

[55] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[56] Catherine Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of 7th IEEE Computer Security Foundations Workshop*, pages 84–89. IEEE, 1994.

[57] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004. Preliminary version in WITS 2002.

[58] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer-Verlag, 2004.

[59] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *Proceedings of Seventh USENIX Security Symposium*, pages 201–216, 1998.

[60] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). *Electr. Notes Theor. Comput. Sci.*, 45, 2001.

[61] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.

[62] John C. Mitchell, Arnab Roy, and Mukund Sundararajan. An Automated Approach for Proving PCL Invariants. *Electr. Notes Theor. Comput. Sci.*, 234: 93-113, 2009.

[63] Dusko Pavlovic and Catherine Meadows. Deriving secrecy properties in key establishment protocols. In *Proceedings of ESORICS*, 2006.

[64] D.H. Phan and David Pointcheval. Une comparaison entre deux methodes de preuve de securite. In *Proc. of RIVF*, pages 105–110, 2003. In French.

[65] Ajith Ramanathan, John C. Mitchell, Andre Scedrov, and Vanessa Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Proceedings*, volume 2987 of *Lecture Notes in Computer Science*, pages 468–483. Springer-Verlag, 2004.

[66] Arnab Roy, Anupam Datta, Ante Derek, and John C. Mitchell. Inductive proofs of computational secrecy. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2007.

[67] Arnab Roy, Anupam Datta, Ante Derek, and John C. Mitchell. Inductive trace properties for computational security. *WITS*, 2007. Full version at http://www.stanford.edu/∼arnab/rddm-IndTraceProps.pdf.

[68] Arnab Roy, Anupam Datta, Ante Derek, John C. Mitchell, and Jean-Pierre Seifert. Secrecy analysis in Protocol Composition Logic., 2006. In *Proceedings of 11th Annual Asian Computing Science Conference*, 2006.

[69] Arnab Roy, Anupam Datta, and John C. Mitchell. Formal proofs of cryptographic security of diffie-hellman-based protocols. In Gilles Barthe and Cédric Fournet, editors, *TGC*, volume 4912 of *Lecture Notes in Computer Science*, pages 312–329. Springer, 2007.

[70] Bogdan Warinschi. A computational analysis of the Needham-Schroeder(-Lowe) protocol. In *Proceedings of 16th Computer Science Foundation Workshop*, pages 248–262. ACM Press, 2003.

[71] L. Zhu and B. Tung. Public key cryptography for initial authentication in kerberos, 2006. Internet Draft.