Inductive Proof Method for Computational Secrecy

Arnab Roy, Anupam Datta, Ante Derek, John C. Mitchell

Department of Computer Science, Stanford University

Abstract. We investigate inductive methods for proving secrecy properties of network protocols, in a "computational" setting applying a probabilistic polynomial-time adversary. As in cryptographic studies, our secrecy properties assert that no probabilistic polynomial-time distinguisher can win a suitable game presented by a challenger. Our method for establishing secrecy properties uses inductive proofs of computational trace-based properties, and axioms and inference rules for relating trace-based properties to non-trace-based properties. We illustrate the method, which is formalized in a logical setting that does not require explicit reasoning about computational complexity, probability, or the possible actions of the attacker, by giving a modular proof of computational authentication and secrecy properties of the Kerberos V5 protocol.

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. Historically, most efforts used an abstract *symbolic model,* also referred to as the *Dolev-Yao* model [30, 32, 23]. 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, 6, 7, 14, 25, 3, 31, 19, 21, 33].

A fundamental problem in reasoning about secrecy, such as computational indistinguishability of the key used in a protocol from a randomly chosen one, 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 paper is a form of trace-based property, called *secretive,* suitable for inductive and compositional proofs, together with a form of standard cryptographic reduction proof which shows that any attack on a secretive protocol yields an attack on cryptographic primitives used in the protocol. This reduction method can be used to prove weaker security properties than indistinguishability, from weaker security assumptions about primitives used in the protocol. We give the cryptographic reduction in a precise form, by inductively

defining the operational behavior of a simulator that simulates the protocol to the protocol adversary, bringing several subtleties to light. An essential problem in defining the simulator, which interacts with cryptographic primitives through the game used to characterize their security, is that the simulator's actions must be uniquely determined, without knowing which of two possible secrets is the one actually used by the protocol. We solve a non-trivial problem associated with protocol actions such as unpairing and decryption by introducing a reasonable type-tagging assumption on the computational implementation of the primitives, allowing the simulator to proceed in these cases.

We leverage the semantic proof that secretive protocols yield black-box reductions by presenting an inductive method for showing that a protocol is secretive, formulated in Computational Protocol Composition Logic (CPCL) [19, 21]. In the process, we generalize a previous induction rule, so that only one core induction principle is needed in the logic. We also extend previous composition theorems [18, 24] 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. An inherent advantage of our approach is that induction proceeds only over action sequences of the protocol program, as executed by honest protocol participants, yet the conclusion is sound for protocol execution in the presence of an arbitrary probabilistic polynomial-time attacker.

Our approach may be compared with equivalence-based methods, such as used in [4] to derive computational properties of Kerberos V5 from a symbolic proof. In equivalence-based methods [6, 31, 16, 26, 5], the behavior of a symbolic abstraction, under symbolic attacks, must yield the same observable behavior as a computational execution under computational (probabilistic polynomial-time) attack. For the standard symbolic model, this appears to require strong cryptographic assumptions, although perhaps weaker cryptographic assumptions can be accommodated by developing new symbolic models. The approach that we advance in this paper involves high-level reasoning methods that do not involve probability or complexity, yet are sound when interpreted over computational protocol execution and attack. In a sense, computational soundness of a symbolic logic only requires an implicational connection between symbolic reasoning and computational execution, whereas equivalence-based approaches require a stronger correspondence. While we believe that both approaches have merit, two specific technical points that distinguish the current state of each are (i) the need to prove the absence of a "commitment problem," in addition to symbolic security, in [4], and (ii) the apparent open problem expressed in [4] of developing compositional methods in that framework.

The Kerberos [28] protocol is widely used for authentication in a variety of settings. The basic protocol has three sections, each involving an exchange between the client and a different service. Our formal proof is modular, with the proof for each section assuming a precondition and establishing a postcondition that implies the precondition of the following section. In addition, the similarities between different two-step sections of the protocol make it possible to construct template proofs that can be reused for each two-step message exchange, further simplifying the development of the proof.

Section 2 describes the protocol process calculus and computational execution model. A trace-based definition of "secretive protocols" and relevant computational notions are explained in section 3. Axioms and proof rules appear in section 4, with composition theorems developed in section 5 and applied in the proofs for Kerberos in section 6. Conclusions appear in section 7.

2 Syntax and Semantics

In this section, we review the relevant parts of the protocol programming language, logic, and security model for key exchange developed in our earlier work [22, 17–19, 21].

2.1 Modeling Protocols

A simple protocol programming language is used to represent a *protocol* by a set of *roles*, such as "Client", or "Server", each specifying a sequence of actions to be executed by an honest participant (see [22, 17, 18]). Protocol actions include nonce generation, encryption, decryption and communication steps (sending and receiving). Every principal can be executing one or more copies of each role at the same time. We use the word *thread* to refer to a principal executing one particular instance of a role. Each *thread* X is identified by a pair $(\hat{X}, \eta)$, where $\hat{X}$ is a principal and η is a unique session identifier.

Syntax To illustrate the syntax of this language, we use it to describe the Kerberos V5 protocol [28]. It involves trusted principals known as the Kerberos Authentication Server (KAS) and the Ticket Granting Server (TGS). There are pre-shared long term keys between the client and the KAS, the KAS and the TGS, and 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. Note that typically the KAS shares long-term keys with a number of clients and the TGS with a number of application servers.

Kerberos has four roles, one for each kind of participant - **Client**, **KAS**, **TGS** and **Server**. The long-term shared keys are written here in the form $k^{type}_{X,Y}$ where X and Y are the principals sharing the key. The *type* appearing in the superscript indicates the relationship between X and Y in the transactions involving the use of the key. There are three *type*s required in Kerberos: $c \rightarrow k$ 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. Kerberos runs in three stages with the client role participating in all three. The description of the roles is based on the A level formalization of Kerberos V5 in [12]. We describe the formalization of the first stage here; the complete formalization is given in Appendix E.

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.

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` $AKey.n_1.\hat{T}$;

... ...

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` $\hat{C}.tgt.enc_{kc}$;

$]_K$

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.

Protocol Execution Model We consider a standard two-phase execution model as in [11]. In the initialization phase of protocol execution, we assign a set of roles to each principal, identify a subset which is honest, and provide all entities with encryption keys and random coins. In the execution phase, the adversary executes the protocol by interacting with 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 [11]. 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.

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 parties as well as the mapping of bitstrings to variables. On the other hand, although the attacker may produce and send arbitrary bitstrings, the trace only records the send-receive actions of the attacker, and not its internal actions. The technical definition of a trace includes, in addition, the random bits used by the honest parties, the adversary and the distinguisher, as well as a few other elements that are used in defining semantics of formulas over traces [19]. In section 3, we omit these additional fields and refer to a trace as $\langle e, \lambda \rangle$, where e is a symbolic description of the trace and λ maps terms in e to bitstrings.

In the computational implementation of protocol actions by honest parties, we require that constructed terms carry certain type information. The type of a term determines the operations that can be applied to it. In particular, nonces, ids and constant strings cannot be unpaired or decrypted, pairs cannot be decrypted, encryptions cannot be unpaired and encryption with one key cannot be decrypted with another. This may, for example, be easily implemented as follows: nonces, ids and constant strings are prefixed with tags indicating their types, pairs are prefixed with a 'pair' tag and encryptions with key k are prefixed with an 'encrypted with key k' tag. However, the adversary is not restricted in the same way–it can modify or spoof tags or produce arbitrary untagged bitrings.

Action Predicates:
$\mathsf{a} ::= \mathsf{Send}(X,t) \mid \mathsf{Receive}(X,t) \mid \mathsf{SymEnc}(X,t,k) \mid \mathsf{SymDec}(X,t,k) \mid \mathsf{New}(X,n)$
Formulas:
$\varphi ::= \mathsf{a} \mid t = t \mid \mathsf{Start}(X) \mid \mathsf{Honest}(\hat{X}) \mid \mathsf{Possess}(X,\ t) \mid \mathsf{Indist}(X,\ t) \mid$
$\quad \mathsf{GoodKeyAgainst}(X,\ t) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists V.\, \varphi \mid \forall V.\, \varphi \mid \neg\varphi \mid \varphi \supset \varphi \mid \varphi \Rightarrow \varphi$
Modal formulas:
$\Psi ::= \varphi\ [Actions]_X\ \varphi$

Table 1. Syntax of the logic

2.2 Computational PCL

Syntax The formulas of the logic are given in Table 1. Protocol proofs usually use modal formulas of the form $\psi[P]_X\varphi$. The informal reading of the modal formula is that if X starts from a state in which ψ holds, and executes the program P, then in the resulting state the security property φ is guaranteed to hold irrespective of the actions of an attacker and other honest principals. Many protocol properties are naturally expressible in this form. Most formulas have the same intuitive meaning as in the symbolic model [17, 18].

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathsf{Send}(X,t)$ holds in a run where the thread X has sent the term t. Action predicates are useful for capturing authentication properties of protocols since they can be used to assert which principals sent and received certain messages. $\mathsf{SymEnc}(X,t,k)$ and $\mathsf{SymDec}(X,t,k)$ respectively mean that X encrypts or decrypts the term t with symmetric key k, while $\mathsf{New}(X,n)$ means X generates fresh nonce n. $\mathsf{Honest}(\hat{X})$ means that the principal $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol. $\mathsf{Start}(X)$ means that the thread X did not execute any actions in the past. $\mathsf{Possess}(X,t)$ means X possesses term t. This is "possess" in the symbolic sense of computing the term t using Dolev-Yao rules. $\mathsf{Indist}(X,t)$ means that agent X cannot tell the bitstring representation of the term t from another bitstring chosen at random from the same distribution. This predicate captures a strong notion of cryptographic secrecy. The logical connectives have standard interpretations. The only exception is the conditional implication ($\Rightarrow$), which is related to a form of conditional probability and appears essential for reasoning about cryptographic reductions (see [19] for further discussion).

Semantics Intuitively, a protocol Q satisfies a formula φ, written $Q \models \varphi$ if for all adversaries and sufficiently large security parameters, the *probability* that φ "holds" is asymptotically close to 1 (in the security parameter). Technically, the probability measure is represented using the cardinality of sets of traces. The meaning 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 example, an action predicate such as Send selects a set of traces in which a send occurs. The semantics $[\![\varphi]\!]\ (T, D, \epsilon)$ of a formula φ is inductively defined 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 and $\mathsf{GoodKeyAgainst}$, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value or winning an IND-CCA game, respectively. $Q \models \varphi$ if for all adversaries A, distinguishers D, and sufficiently large

security parameters η, $[\![\varphi]\!](T, D, \epsilon)$ is an overwhelming subset of the set T of all possible traces produced by the interaction of protocol Q and adversary A. In other words, the probability $\mid [\![\varphi]\!](T, D, \nu(\eta)) \mid / \mid T \mid \geq 1 - \nu(\eta)$, where $\nu(.)$ is a negligible function, i.e., $\nu(.)$ is smaller than the inverse of every polynomial. The precise inductive semantics for formulas is in [20] (see Appendix A for a summary).

In particular, the semantics of the predicate $\mathsf{GoodKeyAgainst}(X, k)$ is defined using a standard cryptographic style game condition. It captures the intuition that a key output by a secure key exchange protocol should be suitable for use in some application protocol of interest (e.g. as a key for an IND-CCA secure encryption scheme) [21]. Formally, $[\![\mathsf{GoodKeyAgainst}(X, k)]\!](T, D, \epsilon)$ is the complete set of traces T if the distinguisher D, who is given X's view of the run has an advantage less than ϵ in winning the IND-CCA game [8] against a challenger using the bitstring corresponding to term k as the key, and $\emptyset$ otherwise. Here the probability is taken by choosing a uniformly random trace $t \in T$ (which includes the randomness of all parties, the attacker and the distinguisher). The same approach can be used to define other game conditions based on the application protocol.

A *trace property* is a formula φ such that for any set of protocol traces T, $[\![\varphi]\!](T) = \bigcup_{t \in T} [\![\varphi]\!](\{t\})$. The distinguisher and tolerance are omitted since they are not used in defining semantics for such predicates. Thus all action formulas, such as $\mathsf{Send}(X, m)$, are trace properties whereas aggregrate properties such as $\mathsf{Indist}(X, k)$ and $\mathsf{GoodKeyAgainst}$ (X, k) are not.

3 Secretive Protocols

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 and $\mathcal{K}$ be the symbolic representations of a nonce and a set of keys associated with a specific thread in a trace $\langle e, \lambda \rangle$. Define $\Lambda(\mathcal{K}) = \{\lambda(k) | k \in \mathcal{K}\}$.

Definition 1 (Secretive Trace). *A trace $\langle e, \lambda \rangle$ 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 a new nonce r in e, with $\lambda(r) = \lambda(s)$, ensures that r is encrypted with a key k with bitstring representation $\lambda(k) \in \Lambda(\mathcal{K})$ in any message sent out.*
- *whenever a thread decrypts a message with a key k with $\lambda(k) \in \Lambda(\mathcal{K})$, 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 $\lambda(k') \in \Lambda(\mathcal{K})$ in any message sent out.*

To lift this definition of secretive traces to secretive protocols we need a way to identify the symbol s and the set of symbols $\mathcal{K}$ in each protocol execution trace. We do this by assuming functions $\bar{s}$ and $\bar{\mathcal{K}}$ that map a trace to symbols in the trace corresponding to s and the set of keys in $\mathcal{K}$ respectively. In applications, these mappings will be induced by logical formulas.

Definition 2 (Secretive Protocol). *Given the mappings $\bar{s}$ and $\bar{\mathcal{K}}$, A protocol $\mathcal{Q}$ is a* secretive protocol *with respect to s and $\mathcal{K}$ if for all probabilistic poly-time adversaries $\mathcal{A}$ and for all sufficiently large security parameters η, the probability that a trace t, generated by the interaction of $\mathcal{A}$ with principals following roles of $\mathcal{Q}$, is a secretive trace with respect to $\bar{s}(t)$ and $\bar{\mathcal{K}}(t)$ is overwhelmingly close to 1, the probability being taken over all adversary and protocol randomness.*

A *level-0* key for a protocol execution is an encryption key which is only used as a key but never as a payload. We use multi-party security definitions due to Bellare, Boldyreva and Micali [8] applied to symmetric encryption schemes in the following theorems. In [8], IND-CCA2 and the multi-party IND-CCA game are shown to be asymptotically equivalent.

In all the proofs to do with secretive protocols, we implicitly look at the subset of all traces that are secretive among all possible traces. Since the set of non-secretive traces is a negligible subset of all traces, adversary advantages retain the same asymptotic behaviour - negligible advantages remain negligible and non-negligible advantages remain non-negligible.

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 multi-party Left-or-Right encryption oracles $\mathcal{E}_{k_i}(LoR\,(\cdot,\cdot,b))$ parameterized by a challenge bit b and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}$ (Following [8], $LoR(m_0, m_1, b)$ 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 randomly chooses two bit-strings s_0, s_1 as alternate representations of the putative secret s and then simulates execution of the protocol to the protocol adversary $\mathcal{A}$ for both the representations.

The operational semantics of the bilateral simulator is outlined in table 3 and is intuitively explained as follows: Every action in each thread is considered one by one, scheduled in the same way as a usual protocol execution. We describe the pairing rule below to illustrate the notations and the rest is similar.

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{m}'' \quad \mathsf{m} := \mathtt{pair}\ \ \mathsf{m}', \mathsf{m}'';}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = pair(lv(\mathsf{m}'), lv(\mathsf{m}'')),\ rv(\mathsf{m}) = pair(rv(\mathsf{m}'), rv(\mathsf{m}''))}$$

The notation $\triangleright\ \mathsf{m}$ means the symbol m has been computationally evaluated according to the semantics. The premise of the rule requires that the symbols m and m' have already been evaluated and we are considering the action $\mathsf{m} := \mathtt{pair}\ \ \mathsf{m}', \mathsf{m}''$. The functions lv and rv map a symbol to its bit-string values corresponding to the representations s_0 and s_1 of s respectively. The function *pair* is the actual computational implemention of pairing. What the conclusion of the rule tells us is that the $lv(\mathsf{m})$ is evaluated by pairing the bit-strings $lv(\mathsf{m}')$ and $lv(\mathsf{m}'')$ and similarly for $rv(\mathsf{m})$.

Suppose m 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 $(lv(m), rv(m))$ to be encrypted by k. In addition, this pair of bitstrings is recorded and the result of the query is logged in the set qdb_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 by accessing the log qdb_k- 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 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 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 s_0. From the operational semantics point of view we will always have $lv(m) = rv(m)$ for any message m being sent out–hence the simulator will not enter a *reject* state due to the send action.

In addition to term constructors, protocols also have deconstructors such as unpairings and decryptions, and pattern matching actions. 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 there is a match for $b = 0$ then there should also be a match for $b = 1$ and similarly for a mismatch; if the term for $b = 0$ can be unpaired or decrypted then the corresponding operation also succeeds for the term for $b = 1$ and vice versa for failure. It turns out that the type information carried by terms (mentioned in section 2.1) ensures this consistency in an overwhelming number of traces. This is stated in theorems 1 and 2.

Theorem 1 (Matching Deconstructions). *If the bitstring representation of the symbol 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 $\triangleright\, m$ and $lv(m) = pair(l_0, l_1)$ for some l_0, l_1, then $rv(m) = pair(r_0, r_1)$ for some r_0, r_1 and vice versa.*
- *If $\triangleright\, m, \triangleright\, k$ and $lv(m) = enc(l, lv(k))$ for some l, then $rv(m) = enc(r, lv(k))$ for some r and vice versa.*

Theorem 2 (Matching Terms). *If the bitstring representations of the symbols m and m' coincide on one side of a bilateral simulation, then with overwhelming probability, they coincide on the other side too. Formally, the event $E = \exists m, m'.\ [\triangleright\, m, \triangleright\, m', (lv(m) = lv(m')) \oplus (rv(m) = rv(m'))]$ occurs with probability negligible in the security parameter.*

Theorem 1 implies that deconstruction actions like unpairing and decryption can be carried out consistently on both sides of the simulation. Theorem 2 states that in all but negligible number of traces, matching actions would have consistent results on both the 'left' and the 'right' sides–either success on both or failure on both. The proofs are in Appendix B.

Theorem 3 (CCA security - level 1). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of level-0 keys $\mathcal{K}$.*

- *In the case that s is never used as a key by the honest principals: the adversary has negligible advantage at distinguishing s from random, after the interaction, if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies key indistinguishability for s.*
- *In the case that honest principals are allowed to use s as a key: the adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key s, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies IND-CCA key usability for s.*

Proof. 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 $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ or an IND-CCA adversary $\mathcal{A}_2$ with non-negligible advantages against the encryption scheme. In the first phase, $\mathcal{A}_1$ provides a bilateral simulation of the protocol to $\mathcal{A}$ using the $|\mathcal{K}|$-IND-CCA oracle according to the operational semantics of table 3.

Case 1 s not used as a key by honest principals: In the second phase, $\mathcal{A}_1$ chooses a bit d' and sends $x_{d'}$ 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:

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

Note that this formulation is equivalent to the expression $2Pr[d = b] - 1$ by algebraic manipulations and the assumption $Pr[b = 0] = Pr[b = 1] = 1/2$.

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.

Case 2 s used as a key by honest principals: In the second phase, $\mathcal{A}_1$ uniformly randomly chooses a bit b' and provides oracles $\mathcal{E}_{s_0}(LoR(\cdot,\cdot,b'))$ 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:

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

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 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 s_1 and has access to an encryption oracle $\mathcal{E}_{s_0}(LoR(\cdot,\cdot,b_1))$ and a decryption oracle $\mathcal{D}_{s_0}(\cdot)$. It has a similar behaviour towards $\mathcal{A}$ as $\mathcal{A}_1$ had except that when constructing terms with s, it uses s_1 but when required to encrypt or decrypt using s, it queries $\mathcal{E}_{s_0}(LoR(\cdot,\cdot,b_1))$ or $\mathcal{D}_{s_0}(\cdot)$. In the second phase, $\mathcal{A}_1$ uses the oracles $\mathcal{E}_{s_0}(LoR(\cdot,\cdot,b_1))$ 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 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:

$$(1/2)\mathbf{Adv}_{IND-CCA,\mathcal{A}_2}(\eta) = Pr[d_1 = b_1] - 1/2 = Pr[d = 0|b = 1] - 1/2 \tag{3}$$

By the equations 2 and 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. □

If a protocol is a *secretive protocol* with respect to nonce k and set of level-0 keys $\mathcal{K}$ then we will call k a level-1 key for the protocol, protected by $\mathcal{K}$. 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 [10] and also referred to as *existential unforgeability* of ciphertexts in [27].

Theorem 4 (CTXT integrity - level 1). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of level-0 keys $\mathcal{K}$. During the protocol run, if an honest principal decrypts a ciphertext with key s successfully, then with overwhelming probability the ciphertext was produced by an honest principal by encryption with s if the encryption scheme is IND-CCA and INT-CTXT secure.*

The proof is outlined in Appendix C. We now extend theorems 3–4 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.

Definition 3 (Key Graph). *Let $\mathcal{K}$ be the symbolic representations of nonces and keys associated with a specific thread in a trace $\langle e, \lambda \rangle$. The* key graph *of $\mathcal{K}$ in a protocol is a directed graph with keys in $\mathcal{K}$ as vertices. There is an edge from key k_1 to k_2 if the protocol is secretive with respect to k_2 and a key set which includes k_1.*

Definition 4 (Key Level). *Consider a directed acyclic key graph. Keys at the root are level 0 keys. The level of any other key is one more than the maximum level among its immediate predecessors.*

Definition 5 (Key Closure). *For a set of keys $\mathcal{K}$ from a directed acyclic key graph, we define its closure $\mathcal{C}(\mathcal{K})$ to be the union of sets of keys at the root which are predecessors of each key in $\mathcal{K}$.*

Theorem 5 (CCA security - Key DAGs). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of keys $\mathcal{K}$ in a DAG of finite and statically bounded level.*

- *In the case that s is never used as a key by the honest principals: the adversary has negligible advantage at distinguishing s from random, after the interaction, if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies key indistinguishability for s.*
- *In the case that honest principals are allowed to use s as a key: the adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key s, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies IND-CCA key usability for s.*

Theorem 6 (CTXT integrity). *Assume that a probabilistic poly-time adversary interacts with a secretive protocol with respect to nonce s and a set of keys $\mathcal{K}$ in a DAG of finite, statically bounded levels. During the protocol run, if an honest principal decrypts a ciphertext with key s successfully, then with overwhelming probability the ciphertext was produced by an honest principal by encryption with s if the encryption scheme is IND-CCA and INT-CTXT secure.*

The proofs are outlined in Appendix C.

4 Proof System

In this section, we present a general induction rule, axiomatize the informal definition of a *secretive protocol* given in section 3 and formulate axioms stating that *secretive protocols* guarantee certain computational properties. The soundness proofs of these axioms rely on the theorems in section 3.

4.1 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. 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 are sound for this semantics:

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

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

Lemma 1. *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* $\triangleright\ m$ *and* $lv(m) = rv(m)$*.*

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. □

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,

$$\mathsf{SendGood}(X, s, \mathcal{K}) \equiv \forall m.\ (\mathsf{Send}(X, m) \supset \mathsf{Good}(X, m, s, \mathcal{K}))$$

$$\mathsf{Secretive}(s, \mathcal{K}) \equiv \forall X.\ (\mathsf{Honest}(\hat{X}) \supset \mathsf{SendGood}(X, s, \mathcal{K}))$$

The axioms **SG0** − **2** are based on the definition of SendGood:

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

SG1 $\mathsf{SendGood}(X, s, \mathcal{K})\ [\mathtt{a}]_X\ \mathsf{SendGood}(X, s, \mathcal{K})$, 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 that consistency of deconstruction is ensured on both sides of the corresponding bilateral simulation - e.g.

unpairing succeeds on one side iff it succeeds on the other. Theorem 1 ensures this consistency. Similarly, soundness for matching actions follow from theorem 2. Soundness of **SG2** follows from the operational semantics of the simulator on a send action and lemma 1.

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 $\mathsf{Secretive}(s, \mathcal{K})$.

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

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

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 $\mathsf{Start}(X)\ []_X\ \Psi(X)$.

4.2 Relating Secretive Protocols to Good Keys

Now we relate the concept of a secretive protocol, which is trace-based, to complexity theoretic notions of security. As defined in section 3, 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.

For a set of keys $\mathcal{K}$ of levels ≤ 1, recall from definition 5 that $\mathcal{C}(\mathcal{K})$ 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 $\mathcal{C}(\mathcal{K})$. $\mathsf{GoodInit}(s, \mathcal{K})$ asserts that all such threads belong to honest principals. Formally,

$$\mathsf{InInitSet}(X, s, \mathcal{K}) \equiv \exists k \in \mathcal{C}(\mathcal{K}).\ \mathsf{Possess}(X, k) \vee \mathsf{New}(X, s)$$

$$\mathsf{GoodInit}(s, \mathcal{K}) \equiv \forall X.\ (\mathsf{InInitSet}(X, s, \mathcal{K}) \supset \mathsf{Honest}(\hat{X}))$$

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 $\mathsf{GoodKeyFor}$ lets us state this. 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 soundness of the following axiom follows from theorems 3 and 5:

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

If the encryption scheme is both IND-CCA and INT-CTXT secure then, the soundness of the following axioms follow from theorems 4 and 6:

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

The proof system is summarized in Appendix D. The **soundness theorem** is proved by showing that every axiom is a valid formula and that all proof rules preserve validity. Proofs for selected axioms are given in Appendix C.

Theorem 7 (Soundness). $\forall \mathcal{Q}, \varphi.$ *if* $\mathcal{Q} \vdash \varphi$ *then* $\mathcal{Q} \vDash \varphi$

5 Compositional Reasoning for Secretive Protocols

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 [18, 24]. 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 [18] 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 6 (Parallel Composition). *The parallel composition $\mathcal{Q}_1 \otimes \mathcal{Q}_2$ 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 8 (Parallel Composition). *If $\mathcal{Q}_1 \vdash \Gamma$ and $\Gamma \vdash \Psi$ and $\mathcal{Q}_2 \vdash \Gamma$ then $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \vdash \Psi$, where Γ denotes the set of invariants used in the proof of Ψ.*

Definition 7 (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. [18]).

Theorem 9 (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)*
   - $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \vdash \theta_{P_1}[P_1]_X\ \theta_{P_2}$
   - $\forall i. \forall S \in BS(P_i).\ \theta_{P_i}[S]_X\ \theta_{P_i}$.

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 8 (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 [28], IEEE 802.11i [1], and IKEv2 [13], 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. [24]). Note that the roles are built up from a finite number of basic sequences of the component protocol roles.

Theorem 10 (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)*
   - $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \cdots \otimes \mathcal{Q}_n \vdash \mathsf{Start}(X) \supset \theta_{P_1}$ *and* $\mathcal{Q}_1 \otimes \mathcal{Q}_2 \cdots \otimes \mathcal{Q}_n \vdash \forall i.\, \theta_{P_i}[P_i]_X\ \theta_{P_{i+1}}$
   - $\forall i. \forall S \in \bigcup_{j \geq i} BS(P_j).\ \theta_{P_i}[S]_X\ \theta_{P_i}$.

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.

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

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

Table 2. Kerberos Security Properties

6 Analysis of Kerberos

Table 2 lists the security properties of Kerberos that we want to 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 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 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 11 (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. Similar result holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash AUTH^{client}_{kas}, AUTH^{tgs}_{kas}$.

Proof Sketch. 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 2), the client decrypts a ciphertext encrypted with a key shared only between itself and the KAS ($k^{c \to k}_{C,K}$). 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 (explained in Appendix D) 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 2). This is also an invariant of Kerberos. We now have a high level intuition of the entire proof.

The formal proof in Appendix F.2 follows this high level intuition. In course of execution of the **Client** role by principal $\hat{C}$, it decrypts the message $E_{sym}[k^{c \to k}_{C,K}](AKey.n_1.\hat{T})$. Using axiom **CTX0**, we derive that it was encrypted by one of the owners of $k^{c \to k}_{C,K}$ - 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 $AUTH^{tgs}_{kas}$ is along identical lines. In Appendix F.2, we first give a *template* proof for the underlying reasoning and then instantiate it for both $AUTH^{client}_{kas}$ and $AUTH^{tgs}_{kas}$. In $AUTH^{tgs}_{kas}$, 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 12 (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,* $KERBEROS \vdash SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$.

Proof Sketch. This theorem states a form of secrecy property for the Authentication Key $AKey$ - specifically, that $AKey$ is good for use as an encryption key in the sense of the security model described in section 2.2. Observe that in the first stage, the KAS sends out $AKey$ encrypted under two different keys - $k^{c \to k}_{C,K}$ and $k^{t \to k}_{T,K}$, 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 $\mathcal{K} = \{k^{c \to k}_{C,K}, k^{t \to k}_{T,K}\}$ 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^{kas}_{akey}), 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^{client}_{akey} and SEC^{tgs}_{akey}), 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, it turns out that our initial conjecture is correct.

In the formal proof in Appendix F.3, 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 (theorem 10) 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 11). 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 3 (CCA security - level 1), which is formally expressed by axiom **GK**.

Theorem 13 (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. Similar result holds for a principal executing the* **Server** *role. Formally,* $KERBEROS \vdash AUTH^{client}_{tgs}, AUTH^{server}_{tgs}$.

Proof Sketch. The proof of $AUTH^{server}_{tgs}$ can be instantiated from the template proof used for theorem 11 and is formally done in Appendix F.2. The proof of $AUTH^{client}_{tgs}$ depends on the 'goodkey'-ness of $AKey$ established by theorem 12 and is much more involved. We omit discussion of the proof due to space constraint but a formal proof is given in Appendix F.4.

Theorem 14 (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. Similar result holds for a principal executing the* **TGS** *role. Formally,* $KERBEROS \vdash SEC^{client}_{skey}, SEC^{tgs}_{skey}$.

Proof Sketch. The idea here is that the Service Key $SKey$ is protected by level-0 key $k^{s \rightarrow t}_{S,T}$ and level-1 key $AKey$. The proof of 'Secretive'-ness proceeds along the same line as for theorem 12 and uses derivations from theorem 13. Then we invoke axiom **GK** for level-2 keys to establish $KERBEROS \vdash SEC^{client}_{skey}, SEC^{tgs}_{skey}$. We omit detailed discussion of this theorem in this paper.

Kerberos with PKINIT In the first stage of Kerberos with PKINIT [34], the KAS establishes the authorization key encrypted with a symmetric key which in turn is sent to the client encrypted with its public key. A fundamental difference in this setting with respect to the purely symmetric key setting is that now we have to consider both public and symmetric keys at level 0. This necessitates a definition and security analysis of a

joint public-symmetric key game. We have formulated such a definition and extended the proof system although due to shortage of space we do not include the results in this paper. The extended proof system lets us 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 paper [15], 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.

7 Conclusion

We formalize reasoning about secrecy by introducing axioms and rules for showing that individual receive-send protocol steps respect secrecy of message parts, and an induction rule for reasoning about arbitrarily many simultaneous protocol sessions. These proof principles are shown sound for a probabilistic polynomial-time semantics of protocol execution and attack.

We use the proof system consisting of computationally sound rules from the literature, combined with new axioms and rules presented here, to prove authentication and secrecy properties of the Kerberos protocol. Our concise, modular proof provides assurance about the correctness of Kerberos, assuming that the cryptographic primitives satisfy the technical conditions identified in this paper.

References

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. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
3. P. Adão, G. Bana, and A. 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. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. 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.
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. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.
7. M. Baudet, V. Cortier, and S. 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.
8. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.
9. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.
10. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, pages 531–545, 2000.
11. M. Bellare and P. 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.
12. F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. Verifying confidentiality and authentication in kerberos 5. In *ISSS*, pages 1–24, 2003.
13. E. C. Kaufman. Internet Key Exchange (IKEv2) Protocol, 2005. RFC.
14. R. Canetti and J. 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.
15. I. Cervesato, A. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key kerberos. Technical report.
16. V. Cortier and B. 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.
17. A. Datta, A. Derek, J. C. Mitchell, and D. 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.
18. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.
19. A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *ICALP*, pages 16–29, 2005.
20. A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. 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, 2005.
21. A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proceedings of 19th IEEE Computer Security Foundations Workshop*, pages 321–334. IEEE, 2006.
22. N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:677–721, 2003.
23. F. J. T. Fábrega, J. C. Herzog, and J. 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.
24. C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of ieee 802.11i and tls. In *ACM Conference on Computer and Communications Security*, pages 2–15, 2005.

25. J. Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, MIT, 2004.
26. R. Janvier, L. Mazare, and Y. 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.
27. J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *FSE*, pages 284–299, 2000.
28. J. Kohl and B. Neuman. The kerberos network authentication service, 1991. RFC.
29. Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.
30. C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of 7th IEEE Computer Security Foundations Workshop*, pages 84–89. IEEE, 1994.
31. D. Micciancio and B. 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.
32. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley Publishing Co., 2000.
33. B. Warinschi. A computational analysis of the Needham-Schroeder(-Lowe) protocol. In *Proceedings of 16th Computer Science Foundation Workshop*, pages 248–262. ACM Press, 2003.
34. L. Zhu and B. Tung. Public key cryptography for initial authentication in kerberos, 2006. Internet Draft.

A Computational Semantics

A.1 Computational Traces

Informally, a *run* is a record of all actions executed by honest principals and the attacker during protocol execution. Since honest principals execute symbolic programs, a run will contain symbolic description of actions executed by honest parties as well as the mapping of bitstrings to variables. A run will also include arbitrary bitstrings that attacker decides to save for the distinguishing phase. Since different coin tosses of the attacker can yield same behavior, we will include the attacker randomness R explicitly in the run. *Computational trace* contains two additional elements: randomness R_T used for testing indistinguishability and mapping σ which keeps track of values assigned to quantified variables in the formula.

During the protocol execution, the adversary $\mathcal{A}$ may record any internal, private message on a special *knowledge tape*. This tape is not read by any participant of the protocol. Its content is used to decide if a given security formula is valid or not. We write $\mathcal{K}$ for the list $[(i_1, m_1), .., (i_n, m_n)]$ of messages m_k that $\mathcal{A}$ writes on its knowledge tape. The messages are indexed by the number i_k of actions already executed when m_k is written. This index is useful to remember a previous state of the knowledge tape.

At the end of the protocol execution, the adversary $\mathcal{A}$ 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 represents 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 $\mathcal{O}$ 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: $Start(...)$ stores the initialization data, and the rest is an ordered list of all exchanged messages and honest participants' internal actions.

Definition 9. *(Computational Traces) Given a protocol Q, an adversary $\mathcal{A}$, 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 $\langle e, \lambda, \mathcal{O}, \mathcal{K}, R\rangle$ where e is the symbolic execution strand, $\lambda : Var(e) \to \{0,1\}^{p(\eta)}$ maps the symbolic terms in e to bitstrings, $\mathcal{O}$ is the pair of integers written on the output tape, and $\mathcal{K}$ 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 $\sigma : Var(\varphi) \to \{0,1\}^{p(\eta)}$, a substitution that maps formula variables to bitstrings. The set of computational traces is*

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

Definition 10. *(Participant's View) Given a protocol Q, an adversary $\mathcal{A}$, a security parameter η, a participant $\tilde{X}$ and a trace $t = \langle e, \lambda, \mathcal{O}, \mathcal{K}, R, R_T, \sigma\rangle \in T_Q(\mathcal{A}, \eta)$, $View_t(\tilde{X})$ 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 $\mathcal{K}$, the messages written on the knowledge tape by the adversary.

The following three definitions are a prelude to setting up a semantics of the predicate Indist(). Informally, based on some trace knowledge K, the distinguisher D is trying to determine which of the two bitstrings corresponds to the symbolic term. One of the bitstrings is going to be an actual bitstring representation of the term in the current run, while the other is going to be a random bitstring of the same structure. The order of the two bitstrings when presented to the distinguisher is the output of an LR Oracle using a random selector bit.

Definition 11. *(LR Oracle) The LR Oracle [9] is used to determine the order in which two bitstrings are presented depending on the value of the selector bit, i.e.* $LR(s_0, s_1, b) = \langle s_b, s_{1-b} \rangle$.

Definition 12. *(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 that of* 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* $u = \{v\}^n_K$*:* $f(\{v\}^n_K, \sigma, r_1; r_2) = \mathcal{E}(f(v, \sigma, r_1), \sigma(K), r_2)$

Definition 13. *(Distinguisher) A distinguisher* D *is an polynomial 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'*.*

In order to define the semantics of the modal operator, we introduce operators Pre and $Post$ on sets of traces. Informally, for a strand P of a thread $\tilde{X}$ and the set of traces T, $Post(T_P)$ is going to correspond to runs from T in which P is a terminating segment of the sequence of actions executed by $\tilde{X}$, while $Pre(T_P)$ is corresponds to runs from T, where $\tilde{X}$ is about to start executing actions in P.

Definition 14. *(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* $\mathcal{K}_{\leq p}$ *is the restriction of the knowledge tape* $\mathcal{K}$ *to messages written before the position* p*. We define the set of traces* $Post(T_P) = \{t[s \leftarrow s_1; s_2, K \leftarrow K_{\leq p_2}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$*.*

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 GoodKeyAgainst is inherently more complex.

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. 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 deal with these issued, semantics of a particular formula will be defined with respect to two distinguishers: one for occurrences with positive polarity, and one for occurrences with negative polarity. In the final definition of formula validity we will universally quantify over all positive distinguishers and existentially quantify over all negative distinguishers.

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.

A.2 Inductive Definition of $[\![\varphi]\!](T, D, \epsilon)$

We inductively define the semantics $[\![\varphi]\!](T, D, \epsilon)$ of a formula φ on the set T of traces, with a pair of distinguishers D and tolerance ϵ. In predicates appearing with positive (resp. negative) polarity D stands for the positive (resp. negative) distinguisher. The distinguishers and tolerance are not used in any of the clauses except for Indist and $\mathsf{GoodKeyAgainst}$. In definition 15 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 $\mathtt{send}\ \ \tilde{Y}, v$ 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 paper, 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 $\lambda(s)_{|\sigma(\tilde{X})} = \mathit{null}$. Intuitively, this set contains traces in which $\tilde{X}$ has executed no actions.
- $[\![\mathsf{Contains}(u, v)]\!](T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that there exists a series of decryptions with $\{\lambda(k) \mid k \in \mathit{Key}\}$ and projections (π_1, π_2) constructing $\sigma(v)$ from $\sigma(u)$. This definition guarantees that the result is the whole set T if v is a symbolic subterm of u.
- $[\![\theta \wedge \varphi]\!](T, D, \epsilon) = [\![\theta]\!](T, D, \epsilon) \cap [\![\varphi]\!](T, D, \epsilon)$.
- $[\![\theta \vee \varphi]\!](T, D, \epsilon) = [\![\theta]\!](T, D, \epsilon) \cup [\![\varphi]\!](T, D, \epsilon)$.
- $[\![\neg\varphi]\!](T, D, \epsilon) = T \setminus [\![\varphi]\!](T, D, \epsilon)$.
- $[\![\exists x.\, \varphi]\!](T, D, \epsilon) = \bigcup_\beta([\![\varphi]\!](T[x \leftarrow \beta], D, \epsilon)[x \leftarrow \sigma(x)])$
  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 $\sigma(u) = \sigma(v)$.

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

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

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 14.

Definition 15. *A protocol Q satisfies a formula φ, written $Q \models \varphi$, if $\forall A$ providing an active protocol adversary, $\forall D_P$ providing a positive probabilistic-polynomial-time distinguisher, $\exists D_N$ providing a negative probabilistic-polynomial-time distinguisher, $\exists \nu$ giving a negligible function, $\exists N$, $\forall \eta \geq N$,*

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

where $D = (D_P, D_N)$ and $[\![\varphi]\!](T, D, \nu(\eta))$ 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 9.

B Operational Semantics of the Simulator

$$\frac{\mathsf{m} \text{ in the static list}}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = rv(\mathsf{m}) = getval()} \qquad \frac{\texttt{receive m;}}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = rv(\mathsf{m}) = recv()}$$

$$\frac{\texttt{new n;} \quad \mathsf{n} \neq \mathsf{s}}{\triangleright\ \mathsf{n},\ lv(\mathsf{n}) = rv(\mathsf{n}) = noncegen()} \qquad \frac{\texttt{new s;}}{\triangleright\ \mathsf{s},\ lv(\mathsf{n}) = s_0,\ rv(\mathsf{n}) = s_1}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{m}'' \quad \mathsf{m} := \texttt{pair}\ \mathsf{m}', \mathsf{m}'';}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = pair(lv(\mathsf{m}'), lv(\mathsf{m}'')),\ rv(\mathsf{m}) = pair(rv(\mathsf{m}'), rv(\mathsf{m}''))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{fst}\ \mathsf{m}';}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = fst(lv(\mathsf{m}')),\ rv(\mathsf{m}) = fst(rv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{snd}\ \mathsf{m}';}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = snd(lv(\mathsf{m}')),\ rv(\mathsf{m}) = snd(rv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{K}}{\begin{array}{l}\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = rv(\mathsf{m}) = \mathcal{E}_k(lv(\mathsf{m}'), rv(\mathsf{m}')),\\ qdb_k \leftarrow qdb_k \cup \{lv(\mathsf{m})\}, dec_k^0(lv(\mathsf{m})) = lv(\mathsf{m}'), dec_k^1(lv(\mathsf{m})) = rv(\mathsf{m}')\end{array}}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{K} \quad lv(\mathsf{m}') \notin qdb_k}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = \mathcal{D}_k(lv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{K} \quad lv(\mathsf{m}') \in qdb_k}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = dec_k^0(lv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{K} \quad rv(\mathsf{m}') \notin qdb_k}{\triangleright\ \mathsf{m},\ rv(\mathsf{m}) = \mathcal{D}_k(rv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{K} \quad rv(\mathsf{m}') \in qdb_k}{\triangleright\ \mathsf{m},\ rv(\mathsf{m}) = dec_k^1(rv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{s};}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = enc(lv(\mathsf{m}'), keygen(s_0)),\ rv(\mathsf{m}) = enc(rv(\mathsf{m}'), keygen(s_0))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{s};}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = dec(lv(\mathsf{m}'), keygen(s_0)),\ rv(\mathsf{m}) = dec(rv(\mathsf{m}'), keygen(s_0))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{n} \quad \mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{n}; \quad \mathsf{n} \notin \mathcal{K} \quad \mathsf{n} \neq \mathsf{s}}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = enc(lv(\mathsf{m}'), keygen(lv(\mathsf{n}))),\ rv(\mathsf{m}) = enc(rv(\mathsf{m}'), keygen(rv(\mathsf{n})))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{n} \quad \mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{n}; \quad \mathsf{n} \notin \mathcal{K} \quad \mathsf{n} \neq \mathsf{s}}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = dec(lv(\mathsf{m}'), keygen(lv(\mathsf{n}))),\ rv(\mathsf{m}) = dec(rv(\mathsf{m}'), keygen(rv(\mathsf{n})))}$$

$$\frac{\triangleright\ \mathsf{m} \quad \triangleright\ \mathsf{m}' \quad \texttt{match m as m}'; \quad (lv(\mathsf{m}) = lv(\mathsf{m}')) \oplus (rv(\mathsf{m}) = rv(\mathsf{m}'))}{state = reject}$$

$$\frac{\triangleright\ \mathsf{m} \quad \texttt{send m;} \quad lv(\mathsf{m}) \neq rv(\mathsf{m})}{state = reject}$$

Table 3. Operational Semantics of the Simulator with Parameters $\mathsf{s}, \mathcal{K}$

$$\frac{\texttt{new n;} \quad \mathsf{n} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{C}(\mathcal{K}))}{\triangleright\ \mathsf{n},\ lv(\mathsf{n}) = rv(\mathsf{n}) = noncegen()}$$

$$\frac{\texttt{new n;} \quad \mathsf{n} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{C}(\mathcal{K}))}{\triangleright\ \mathsf{n},\ lv(\mathsf{n}) = noncegen(),\ rv(\mathsf{n}) = noncegen()}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{enc}\ \ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{C}(\mathcal{K})}{\begin{array}{l}\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = rv(\mathsf{m}) = \mathcal{E}_k(lv(\mathsf{m}'), rv(\mathsf{m}')), \\ qdb_k \leftarrow qdb_k \cup \{lv(\mathsf{m})\}, dec_k^0(lv(\mathsf{m})) = lv(\mathsf{m}'), dec_k^1(lv(\mathsf{m})) = rv(\mathsf{m}')\end{array}}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{C}(\mathcal{K}) \quad lv(\mathsf{m}') \notin qdb_k}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = \mathcal{D}_k(lv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{C}(\mathcal{K}) \quad lv(\mathsf{m}') \in qdb_k}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = dec_k^0(lv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{C}(\mathcal{K}) \quad rv(\mathsf{m}') \notin qdb_k}{\triangleright\ \mathsf{m},\ rv(\mathsf{m}) = \mathcal{D}_k(rv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \mathsf{m} := \texttt{dec}\ \ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{C}(\mathcal{K}) \quad rv(\mathsf{m}') \in qdb_k}{\triangleright\ \mathsf{m},\ rv(\mathsf{m}) = dec_k^1(rv(\mathsf{m}'))}$$

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{n} \quad \mathsf{m} := \texttt{enc}\ \ \mathsf{m}', \mathsf{n}; \quad \mathsf{n} \notin \mathcal{C}(\mathcal{K})}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = enc(lv(\mathsf{m}'), keygen(lv(\mathsf{n}))),\ rv(\mathsf{m}) = enc(rv(\mathsf{m}'), keygen(lv(\mathsf{n})))} \quad (*)$$

$$\frac{\triangleright\ \mathsf{m}' \quad \triangleright\ \mathsf{n} \quad \mathsf{m} := \texttt{dec}\ \ \mathsf{m}', \mathsf{n}; \quad \mathsf{n} \notin \mathcal{C}(\mathcal{K})}{\triangleright\ \mathsf{m},\ lv(\mathsf{m}) = dec(lv(\mathsf{m}'), keygen(lv(\mathsf{n}))),\ rv(\mathsf{m}) = dec(rv(\mathsf{m}'), keygen(lv(\mathsf{n})))} \quad (*)$$

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

Table 4. Extension and modification of the Operational Semantics with Parameters $\mathsf{s}, \mathcal{K}$ for Key DAGs

Definition 16 ($\cong$). *For symbols* 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. *The following hold for any bilateral simulation with level-1 keys:*

- *If* $\triangleright\ \mathsf{m}$ *and* $lv(\mathsf{m}) = pair(l_0, l_1)$ *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* $\triangleright\ \mathsf{m}$, $\triangleright\ \mathsf{k}$ *and* $lv(\mathsf{m}) = enc(l, lv(\mathsf{k}))$ *for some* l*, then* $rv(\mathsf{m}) = enc(r, lv(\mathsf{k}))$ *for some* r *and vice versa.*
- *If* $\triangleright\ \mathsf{m}$ *and* $lv(\mathsf{m})$ *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, if* $\triangleright\ \mathsf{m}$ *then* $lv(\mathsf{m}) = rv(\mathsf{m})$

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

- If $\triangleright\ \mathsf{m}$ and $lv(\mathsf{m}) = pair(l_0, l_1)$ 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 $\triangleright\ \mathsf{m}$, $\triangleright\ \mathsf{k}$ and $lv(\mathsf{m}) = enc(l, lv(\mathsf{k}))$ for some l, then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or there exists m', k' such that $\mathsf{m} \cong \mathsf{m}', \mathsf{k} \cong \mathsf{k}'$ and m' is derived by an $\mathtt{enc}\ \cdot, \mathsf{k}'$ action.
– If $\triangleright\ \mathsf{m}$ and $lv(\mathsf{m})$ 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, if $\triangleright\ \mathsf{m}$ then $lv(\mathsf{m}) = rv(\mathsf{m})$

Theorem 15 (Matching Nonces - No Keying - Level 1). *Let E be the event $[\triangleright\ \mathsf{m}\ lv(\mathsf{m}) = rv(\mathsf{m}) = s_0]$. $\Pr[E]$ is negligible in the security parameter η if the encryption scheme is IND-CCA secure.*

Proof. Let $E_i = E \wedge b = i$ for $i \in \{0, 1\}$. Assume on the contrary that $\Pr[E]$ is non-negligible in η. We will construct a $|\mathcal{K}| - IND - CCA$ adversary in the following way:

If E occurs then output $b' = 0$ else output b' uniformly randomly from $\{0, 1\}$.

Now, probability of winning the IND-CCA challenge is:

$$\begin{aligned}\Pr[b' = b] &= \Pr[b = 0].\Pr[b' = 0|b = 0] + \Pr[b = 1].\Pr[b' = 1|b = 1] \\ &= \frac{1}{2}.\left[\frac{|E_0|}{|b = 0|} + \frac{1}{2}.\left(1 - \frac{|E_0|}{|b = 0|}\right)\right] + \frac{1}{2}.\left[\frac{1}{2}.\left(1 - \frac{|E_1|}{|b = 1|}\right)\right] \\ &= \frac{1}{2} + \frac{1}{4}.\left[\frac{|E_0|}{|b = 0|} - \frac{|E_1|}{|b = 1|}\right] \qquad (*)\end{aligned}$$

By assumption $\frac{|E_0|+|E_1|}{|b=0|+|b=1|}$ is non-negligible. We will prove that $\frac{|E_1|}{|b=1|}$ is negligible, thereby implying $\frac{|E_0|}{|b=0|}$ is non-negligible, which together would imply, by (*), that the probability of winning the IND-CCA challenge is non-negligibly over half.

Given $b = 1$, the '1-world' is a consistent simulation with s_1 as the secret with no messages constructed from s_0. Therefore, this side is information theoretically unaware of s_0 and hence the probability that $rv(\mathsf{m}) = s_0$ is bounded by $\frac{\text{max length of a trace}}{2^{\text{length of a nonce}}}$. Now, maximum length of a trace is polynomially bounded in η and length of a nonce is a polynomial in η. Therefore this is a negligible quantity and hence we are done. □

Theorem 16 (Matching Nonces - Keying - Level 1). *Let E be the event $[\triangleright\ \mathsf{m}\ lv(\mathsf{m}) = rv(\mathsf{m}) = s_0]$. $\Pr[E]$ is negligible in the security parameter η if the encryption scheme is IND-CCA secure.*

Proof. Let $E_i = E \wedge b = i$ for $i \in \{0, 1\}$. Assume on the contrary that $\Pr[E]$ is non-negligible in η. We will construct a $|\mathcal{K}|$-IND-CCA adversary in the following way:

If E occurs then output $b' = 0$ else output b' uniformly randomly from $\{0, 1\}$.

Now, probability of winning the IND-CCA challenge is:

$$\begin{aligned}\Pr[b' = b] &= \Pr[b = 0].\Pr[b' = 0|b = 0] + \Pr[b = 1].\Pr[b' = 1|b = 1] \\ &= \frac{1}{2}.\left[\frac{|E_0|}{|b = 0|} + \frac{1}{2}.\left(1 - \frac{|E_0|}{|b = 0|}\right)\right] + \frac{1}{2}.\left[\frac{1}{2}.\left(1 - \frac{|E_1|}{|b = 1|}\right)\right] \\ &= \frac{1}{2} + \frac{1}{4}.\left[\frac{|E_0|}{|b = 0|} - \frac{|E_1|}{|b = 1|}\right] \qquad (*)\end{aligned}$$

By assumption $\frac{|E_0|+|E_1|}{|b=0|+|b=1|}$ is non-negligible. We will prove that $\frac{|E_1|}{|b=1|}$ is negligible, thereby implying $\frac{|E_0|}{|b=0|}$ is non-negligible, which together would imply, by (*), that the probability of winning the IND-CCA challenge is non-negligibly over half.

Given $b = 1$, the '1-world' is a simulation with s_1 as the secret with s_0 used only as a key. Assume $\frac{|E_1|}{|b=1|}$ is non-negligible. We will construct an IND-CCA adversary against a challenger using $keygen(s_0)$ as a key with the assumption that the probability distribution of $keygen(nonce)$ is the same as the probability distribution of the key space. Consider an execution of the same protocol, only using the rv's, which represents the secret s by the bitstring s_1 and calls the oracles $\mathcal{E}_{s_0}(\cdot,\cdot,b)$ and $\mathcal{D}_{s_0}(\cdot)$ for encryptions and decryptions. Since $\frac{|E_1|}{|b=1|}$ is non-negligible, we must have the event $[\triangleright\ \mathsf{m}$ `match m as s`; $rv(\mathsf{m}) = s_0]$ occuring non-negligibly often. Therefore, the actual key s_0 is revealed non-negligibly often. That $rv(\mathsf{m})$ for some such m is actually s_0 can be confirmed at the end by encrypting some string by using the oracle and attempting to decrypt it with $rv(\mathsf{m})$. Since only a polynomially many m are derived this is doable in poly-time. Once the actual key is revealed, the IND-CCA game can be easily won. □

Theorem 17 (Matching Nonces - No Keying). *Let E be the event* $[\triangleright\ \mathsf{m} \quad lv(\mathsf{m}) = rv(\mathsf{m}) = s_0]$. $\Pr[E]$ *is negligible in the security parameter η if the encryption scheme is IND-CCA secure.*

C Proofs of Some Theorems

Proof of theorem 4. 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 $\mathcal{K}$. Suppose during the protocol run, an honest party decrypts a ciphertext with key s successfully which was not produced by an honest party by encryption with 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 3. However, this new $\mathcal{A}_1$ computes d in a different way. Recall that $\mathcal{A}_1$ uses x_0 when it intends to encrypt or decrypt using s. In the course of interaction with $\mathcal{A}$, if $\mathcal{A}_1$ succeeds in decrypting a ciphertext with key x_0 which was not produced at a previous stage by $\mathcal{A}_1$ by encryption with x_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:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \tag{4}$$

Now, $Pr[d = 0|b = 0]$ is the probability of $\mathcal{A}_1$ succeeding in decrypting a ciphertext with key 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-0 key x_0 (as in this case x_0 was only used as a key). Therefore, using a similar idea as proof of theorem 3 we can build an INT-CTXT adversary $\mathcal{A}_2$ against x_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.

Proof of theorem 5. We will prove this by induction over the maximum level of the DAG of $\mathcal{K}$. If $\mathcal{K}$ consists only of level 0 keys then the result follows from theorem 3. Suppose the maximum level in the DAG of $\mathcal{K}$ is $(n + 1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the closure $\mathcal{C}(\mathcal{K})$ of the set of keys $\mathcal{K}$.

Assume that a probabilistic poly-time adversary $\mathcal{A}$ interacts with a secretive protocol with respect to nonce s and the set of keys $\mathcal{K}$. We will show that if $\mathcal{A}$ has non-negligible advantage at 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 multi-party LoR encryption oracles $\mathcal{E}_{k_i}(LoR(\cdot,\cdot,b))$ 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, $\mathcal{A}_1$ chooses random values x_0^i, x_1^i and for s, $\mathcal{A}_1$ chooses random values x_0, x_1. $\mathcal{A}_1$ constructs messages to be sent to $\mathcal{A}$ as follows:

- to encrypt the term $f(s, s^1, s^2, ...)$ with $k_i \in \mathcal{K}'$, use response to oracle query $\mathcal{E}_{k_i}(f(x_0, x_0^1, x_0^2, ...), f(x_1, x_1^1, x_1^2, ...), b)$.
- to encrypt $f(s, s^1, s^2, ...)$ with s^i, use $\mathcal{E}_{x_0^i}(f(x_0, x_0^1, x_0^2, ...))$.

Decryption operations are served analogously.

In the second phase, $\mathcal{A}_1$ chooses a bit d' and sends $x_{d'}$ 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:

$$\begin{aligned}\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) &= Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \\ &= (Pr[d = 0|b = 0] - 1/2) + (Pr[d = 1|b = 1] - 1/2)\end{aligned} \tag{5}$$

The first probability in the RHS is precisely the probability of $\mathcal{A}$ breaking the indistinguishability of x_0 or equivalently of s. In the case when $b = 1$, the terms were constructed in the following manner:

- encrypt $f(s, s^1, s^2, ...)$ with $k_i \in \mathcal{K}'$: $\mathcal{E}_{k_i}(f(x_1, x_1^1, x_1^2, ...))$.
- encrypt $f(s, s^1, s^2, ...)$ with s^i: $\mathcal{E}_{x_0^i}(f(x_0, x_0^1, x_0^2, ...))$.

We observe here that $\mathcal{A}_1$ simulated the execution of another secretive protocol $\mathcal{G}'$ with keys of level $\leq n$ - $x_0^1, x_0^2, ...$ protecting x_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 removed, and hence of maximum level one less. Therefore, we have:

$$Pr[d = 1|b = 1] - 1/2 = (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta) \tag{6}$$

By the equations 5 and 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.

Proof of theorem 5. We will again prove this by induction over the maximum level of the DAG of $\mathcal{K}$. If $\mathcal{K}$ consists only of level 0 keys then the result follows from theorem 3. Suppose the maximum level in the DAG of $\mathcal{K}$ is $(n+1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the closure $\mathcal{C}(\mathcal{K})$ of the set of keys $\mathcal{K}$.

Assume that a probabilistic poly-time adversary $\mathcal{A}$ interacts with a secretive protocol with respect to nonce s and 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 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 3 to construct the adversary $\mathcal{A}_1$. The only additional operation is that to encrypt or decrypt the term m with s, we use x_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 $\mathcal{E}_{x_0}(m_{b'})$. Decryption requests are also served by decrypting with key x_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:

$$\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \tag{7}$$

The first probability is precisely the probability of $\mathcal{A}$ breaking the 'good-key'-ness of x_0 or equivalently of s. In the case when $b = 1$, the terms were constructed in the following manner:

- encrypt $f(s, s^1, s^2, ...)$ with $k_i \in \mathcal{K}'$: $\mathcal{E}_{k_i}(f(x_1, x_1^1, x_1^2, ...))$.
- encrypt $f(s, s^1, s^2, ...)$ with s^i: $\mathcal{E}_{x_0^i}(f(x_0, x_0^1, x_0^2, ...))$.
- encrypt term m with s: $\mathcal{E}_{x_0}(m)$.

We observe here that $\mathcal{A}_1$ simulated the execution of another secretive protocol $\mathcal{G}'$ with keys of level $\leq n$ - $x_0^1, x_0^2, ...$ protecting x_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 removed, and hence of maximum level one less. Therefore, we have:

$$Pr[d = 0|b = 1] - 1/2 = (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta) \tag{8}$$

By the equations 7 and 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.

Proof of theorem 6. We will prove this by induction over the maximum level of the DAG of $\mathcal{K}$. If $\mathcal{K}$ consists only of level 0 keys then the result follows from theorem 4. Suppose the maximum level in the DAG of $\mathcal{K}$ is $(n + 1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the closure $\mathcal{C}(\mathcal{K})$ of the set of keys $\mathcal{K}$. Suppose during the protocol run, an honest party decrypts a ciphertext with key s successfully which was not produced by an honest party by encryption with s.

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 $\mathcal{A}_1$ succeeds in decrypting a ciphertext with key x_0 which was not produced at a previous stage by $\mathcal{A}_1$ by encryption with x_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:

$$\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \tag{9}$$

Now, $Pr[d = 0|b = 0]$ is the probability of $\mathcal{A}_1$ succeeding in producing a ciphertext with key 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 x_0 (Same argument as in proof of theorem 5 - the DAG reduces by one level) which was not produced by encryption with x_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.

Generalized Induction Principle. We present a generalized induction rule **IND** below. Two instances of this rule—$\mathbf{IND}_{GOOD}$, described in the next subsection, and the **HON** rule, developed in prior work (see Appendix D)—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 $\forall \rho \in \mathcal{Q}.\, \forall P \in BS(\rho).\, \psi(X)\ [P]_X\ \Phi \supset \psi(X)$ 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 2.1)

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.

Proof of theorem 7 (Soundness). 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:

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

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 the predicate $\mathsf{SendGood}$.

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 3 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 [10] 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 4 and 6.

D Summary of Definitions, Axioms and Rules

D.1 Summary of New Definitions, Axioms and Rules

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

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

D.2 A Fragment of the Proof System from Earlier Papers

A representative fragment of the axioms and inference rules in the proof system are collected in Table 5. 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 decomposing messages while the encryption axioms symbolically model encryption. The generic rules are used for manipulating modal formulas.

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	$\mathsf{Receive}(X,p(x))[\texttt{match}\ q(x)\ \texttt{as}\ q(t)]_X\ \mathsf{Receive}(X,p(t))$
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 5. Fragment of the Proof System

D.3 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 [29]. 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.

$$\frac{[\,]_X\ \phi \quad \forall \rho \in \mathcal{Q}. \forall P \epsilon BS(\rho).\ \phi\ [P]_X\ \phi}{\mathcal{Q} \vdash \mathsf{Honest}(\hat{X}) \supset \phi}\ \textbf{HON}$$

no free variable in ϕ except X bound in $[P]_X$

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.

E Formal Description of Kerberos

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};
  $]_S$

F Proof of Kerberos Security Properties

F.1 Environmental Assumptions

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.

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

In the subsequent proofs, we will mainly use the following axioms and the honesty rule (all described in Appendix D):

A1 $\mathsf{New}(X, n) \wedge \mathsf{New}(Y, n) \supset X = Y$

A2 $\mathsf{SymEnc}(X, m, k) \supset \mathsf{Possess}(X, k) \wedge \mathsf{Possess}(X, m)$

F.2 Proofs of $AUTH^{client}_{kas}$, $AUTH^{tgs}_{kas}$ 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 $E_{sym}[k^{type}_{X,Y}](M)^r$. 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.

$\mathsf{Hon}(\hat{X}, \hat{Y}), \Gamma_0$	$\mathsf{GoodKey}(k^{type}_{X,Y})$	(1)
Hypothesis	$[\mathbf{Role}]_X\ \mathsf{SymDec}(X, E_{sym}[k^{type}_{X,Y}](M), k^{type}_{X,Y})$	(2)
$\mathbf{CTX0}, (-2, -1)$	$[\mathbf{Role}]_X\ \exists Z.\ \mathsf{SymEnc}(Z, M, k^{type}_{X,Y})$	(3)
Inst $Z \mapsto Z_0, (-1)$	$[\mathbf{Role}]_X\ \mathsf{SymEnc}(Z_0, M, k^{type}_{X,Y})$	(4)
$\mathbf{A2}, (-1)$	$[\mathbf{Role}]_X\ \mathsf{Possess}(Z_0, k^{type}_{X,Y})$	(5)
$\mathsf{Hon}(\hat{X}, \hat{Y}), \Gamma_0, (-1)$	$[\mathbf{Role}]_X\ \hat{Z}_0 = \hat{X} \vee \hat{Z}_0 = \hat{Y}$	(6)
$(-4, -1)$	$[\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{X}, \eta), M, k^{type}_{X,Y})$ $\vee\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y})$	(7)

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

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

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

HON	$\mathsf{Honest}(\hat{X_0}) \wedge \hat{X_0} \neq \hat{Y_0} \supset \neg\mathsf{SymEnc}(X_0, M_0, k^{type}_{X_0,Y_0})$	(9)

$\mathsf{Hon}(\hat{X}), (-1)$	$[\mathbf{Role}]_X\ \neg\exists \eta.\ \mathsf{SymEnc}((\hat{X}, \eta), M, k^{type}_{X,Y})$	(10)
$(-4, -1)$	$[\mathbf{Role}]_X\ \exists \eta.\ \mathsf{SymEnc}((\hat{Y}, \eta), M, k^{type}_{X,Y})$	(11)

Instantiating for $AUTH_{kas}^{client}$:

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

$$\begin{aligned} \mathbf{HON} \quad & \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, (Key, n, \hat{T}_0), k_{C_0,X}^{c \to k}) \\ & \supset \mathsf{Send}(X, \hat{C}_0, E_{sym}[k_{T_0,X}^{t \to k}](Key, \hat{C}_0), E_{sym}[k_{C_0,X}^{c \to k}](Key, n, \hat{T}_0)) \end{aligned} \tag{13}$$

$$\begin{aligned} \mathsf{Hon}(\hat{K}), (-2,-1) \quad & [\mathbf{Client}]_C \; \exists \eta.\, \mathsf{Send}((\hat{K}, \eta), \hat{C}, E_{sym}[k_{T,K}^{t \to k}](AKey, \hat{C}), \\ & E_{sym}[k_{C,K}^{c \to k}](AKey, n_1, \hat{T})) \end{aligned} \tag{14}$$

$$(-1) \quad AUTH_{kas}^{client} \tag{15}$$

Instantiating for $AUTH_{kas}^{tgs}$:

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

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

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

$$(-1) \quad AUTH_{kas}^{tgs} \tag{19}$$

Instantiating for $AUTH_{tgs}^{server}$:

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

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

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

$$(-1) \quad AUTH_{tgs}^{server} \tag{23}$$

F.3 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 ($\mathbf{G}*$) 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 $[\mathbf{Server}]_{S'}$: 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 $\mathbf{G}*$ 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 $[\mathbf{Client}]_{C'}$. 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:

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

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

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

Now we present the formal proof of goodness:

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

$$[\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n'_1) \wedge \mathsf{Send}(C', \hat{C}'.\hat{T}'.n'_1) \quad (1)$$
$$\Phi_1, (-1) \quad [\mathbf{Client}_1]_{C'}\ n'_1 \neq AKey \quad (2)$$
$$\mathbf{G1}, \mathbf{G5}, (-1) \quad [\mathbf{Client}_1]_{C'}\ \mathsf{Good}(C', \hat{C}'.\hat{T}'.n'_1, AKey, \mathcal{K}) \quad (3)$$
$$\mathbf{SG1\text{-}2}, (-1) \quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \quad (4)$$

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

$$\mathbf{SG1} \quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \quad (5)$$

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

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

$$[\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n_2') \wedge \mathsf{Send}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2') \quad (6)$$

$\Phi_1, (-1)$ $\quad [\mathbf{Client}_3]_{C'}\ n_2' \neq AKey \quad (7)$

$\mathbf{G1}, (-1)$ $\quad \theta_3\ [\texttt{new}\ n_2';]_{C'}\ \mathsf{Good}(C', tgt', AKey, \mathcal{K}) \wedge \mathsf{Good}(C', n_2', AKey, \mathcal{K})$ (8)

$\mathbf{G}*, (-1)$ $\quad \theta_3\ [\mathbf{Client}_3]_{C'}\ \mathsf{Good}(C', tgt'.enc_{ct}'.\hat{C}'.\hat{S}'.n_2', AKey, \mathcal{K}) \quad (9)$

$\mathbf{SG1\text{-}2}, (-1)$ $\quad \theta_3 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K})$ (10)

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

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

$text_{tc}' := \texttt{symdec}\ enc_{tc}', AKey'; \texttt{match}\ text_{tc}'\ \texttt{as}\ SKey'.n_2'.\hat{S}';]_{C'}$

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (11)

Precondition $\theta_5 : \mathsf{Good}(C', st', AKey, \mathcal{K})$

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

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

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

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

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (12)

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

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

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

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (13)

Let, $[\mathbf{KAS}]_{K'} : [\texttt{receive}\ \hat{C}'.\hat{T}'.n_1';$

$\texttt{new}\ AKey';$

$tgt' := \texttt{symenc}\ AKey'.\hat{C}', k_{T',K'}^{t\rightarrow k};$

$enc_{kc}' := \texttt{symenc}\ AKey'.n_1'.\hat{T}', k_{C',K'}^{c\rightarrow k};$

$\texttt{send}\ \hat{C}'.tgt'.enc_{kc}';]_{K'}$

Case 1 : $AKey' = AKey$

$[\mathbf{KAS}]_{K'}\mathsf{New}(K', AKey) \wedge \mathsf{SymEnc}(K', AKey.n_1'.\hat{T}', k_{C',K'}^{c\rightarrow k})$ (14)

$\Phi_2, (-1)$ $[\mathbf{KAS}]_{K'} \hat{C}' = \hat{C} \wedge \hat{K}' = \hat{K} \wedge \hat{T}' = \hat{T}$ (15)

(-1) $[\mathbf{KAS}]_{K'} k^{c \to k}_{C',K'} \in \mathcal{K} \wedge k^{t \to k}_{T',K'} \in \mathcal{K}$ (16)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(K', \hat{C}'.tgt'.enc'_{kc}, AKey, \mathcal{K})$ (17)

Case 2 : $AKey' \neq AKey$

$\mathbf{G2}$ $[\texttt{receive}\ \hat{C}'.\hat{T}'.n'_1;]_{K'}\ \mathsf{Good}(K', \hat{C}'.\hat{T}'.n'_1, AKey, \mathcal{K})$ (18)

$\mathbf{G6}, (-1)$ $[\texttt{receive}\ \hat{C}'.\hat{T}'.n'_1;]_{K'}\ \mathsf{Good}(K', n'_1, AKey, \mathcal{K})$ (19)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(\hat{C}'.tgt'.enc'_{kc}, AKey, \mathcal{K})$ (20)

$\mathbf{SG1\text{-}2}, (-4, -1)$ $\mathsf{SendGood}(K', AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SendGood}(K', AKey, \mathcal{K})$ (21)

Let, $[\mathbf{TGS}]_{T'}$: $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;$
$text'_{ct1} := \texttt{symdec}\ enc'_{ct1}, k^{t \to k}_{T',K'};$
$\texttt{match}\ text'_{ct1}\ \texttt{as}\ AKey'.\hat{C}';$
$text'_{ct2} := \texttt{symdec}\ enc'_{ct2}, AKey';$
$\texttt{match}\ text'_{ct2}\ \texttt{as}\ \hat{C}';$
$\texttt{new}\ SKey';$
$st' := \texttt{symenc}\ SKey'.\hat{C}', k^{s \to t}_{S',T'};$
$enc'_{tc} := \texttt{symenc}\ SKey'.n'_2.\hat{S}', AKey';$
$\texttt{send}\ \hat{C}'.st'.enc'_{tc};]_{T'}$

$\mathbf{G2}, \mathbf{G6}$ $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;]_{T'}\ \mathsf{Good}(T', n'_2, AKey, \mathcal{K})$ (22)

$[\mathbf{TGS}]_{T'}\ \mathsf{New}(T', SKey') \wedge \mathsf{SymEnc}(T', SKey'.\hat{C}', k^{s \to t}_{S',T'})$ (23)

$\Phi_3, (-1)$ $[\mathbf{TGS}]_{T'}\ SKey' \neq AKey$ (24)

$\mathbf{G1}, (-1)$ $[\cdots; \texttt{new}\ SKey';]_{T'}\ \mathsf{Good}(T', SKey', AKey, \mathcal{K})$ (25)

$\mathbf{G}*, (-4, -1)$ $[\mathbf{TGS}]_{T'}\ \mathsf{Good}(T', \hat{C}'.st'.enc'_{tc}, AKey, \mathcal{K})$ (26)

$\mathbf{SG1\text{-}2}, (-1)$ $\mathsf{SendGood}(T', AKey, \mathcal{K})\ [\mathbf{TGS}]_{T'}\ \mathsf{SendGood}(T', AKey, \mathcal{K})$ (27)

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: $SKey' \in \mathcal{K}$

G7, (-1) $[\cdots; enc'_{sc} := \texttt{symenc}\ t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (28)

Case 2: $SKey' \notin \mathcal{K}$ (29)

G2, **G6** $[\texttt{receive}\ enc'_{cs1}.enc'_{cs2};]_{S'}\ \mathsf{Good}(S', enc'_{cs2}, AKey, \mathcal{K})$ (30)

G6, **G8**, (-1) $[\cdots; text'_{cs2} := \texttt{symdec}\ enc'_{cs2}, SKey'; \texttt{match}\ enc'_{cs2}\ \texttt{as}\ \hat{C}'.t';]_{S'}$
$\mathsf{Good}(S', t', AKey, \mathcal{K})$ (31)

G7, (-1) $[\cdots; enc'_{sc} := \texttt{symenc}\ t', SKey';]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (32)

$(-4, -1)$ $[\mathbf{Server}]_{S'}\ \mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (33)

SG1-2, (-1) $\mathsf{SendGood}(S', AKey, \mathcal{K})\ [\mathbf{Server}]_{S'}\ \mathsf{SendGood}(S', AKey, \mathcal{K})$ (34)

Theorem 10 $\Phi \supset \mathsf{Secretive}(AKey, \mathcal{K})$ (35)

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

$$KERBEROS \vdash [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$
$$KERBEROS \vdash [\mathbf{KAS}]_K\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$
$$KERBEROS \vdash [\mathbf{TGS}]_T\ \mathsf{Hon}(\hat{C}, \hat{K}, \hat{T}) \supset \Phi$$

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_2 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:

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

F.4 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.

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[AKey](SKey.n_2.\hat{S}), AKey) \tag{1}$$

$\mathbf{CTXL}, (-1, 1)$ $$[\mathbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, SKey.n_2.\hat{S}, AKey) \tag{2}$$

Inst $X \mapsto X_0, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{SymEnc}(X_0, SKey.n_2.\hat{S}, AKey) \tag{3}$$

(-1) $$[\mathbf{Client}]_C\ \mathsf{Possess}(X_0, AKey) \tag{4}$$

$SEC_{AKey}^{client}, (-1)$ $$\hat{X_0} = \hat{C} \wedge \hat{X_0} = \hat{K} \wedge \hat{X_0} = \hat{T} \tag{5}$$

$\mathbf{HON}$ $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key'.n.\hat{S_0}, Key) \wedge Key \neq k_{Z,X}^{c \to k} \supset \exists \hat{K_0}, \hat{C_0}.\ \mathsf{SymDec}(X, E_{sym}[k_{X,K_0}^{t \to k}](Key.\hat{C_0}), k_{X,K_0}^{t \to k}) \wedge \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{S_0,X}^{s \to t}](Key'.\hat{C_0}).E_{sym}[Key](Key'.n.\hat{S_0})) \tag{6}$$

Inst, $(-4, -1)$ $$[\mathbf{Client}]_C\ \mathsf{SymDec}(X_0, E_{sym}[k_{X_0,K_0}^{t \to k}](AKey.\hat{C_0}), k_{X_0,K_0}^{t \to k}) \wedge \mathsf{Send}(X_0, \hat{C_0}.E_{sym}[k_{S,X_0}^{s \to t}](SKey.\hat{C_0}).E_{sym}[AKey](SKey.n_2.\hat{S})) \tag{7}$$

$\mathbf{CTX0}, (-1)$ $$[\mathbf{Client}]_C\ \exists Y.\ \mathsf{SymEnc}(Y, AKey.\hat{C_0}, k_{X_0,K_0}^{t \to k}) \tag{8}$$

Inst $Y \mapsto Y_0, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{SymEnc}(Y_0, AKey.\hat{C_0}, k_{X_0,K_0}^{t \to k}) \tag{9}$$

$\mathbf{A2}, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{Possess}(Y_0, AKey) \tag{10}$$

$SEC_{AKey}^{client}, (-1)$ $$\mathsf{Honest}(\hat{Y_0}) \tag{11}$$

$\mathbf{HON}$ $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{W}, k_{X,Z}^{t \to k}) \supset \mathsf{New}(X, Key) \tag{12}$$

$(-4, -1)$ $$[\mathbf{Client}]_C\ \mathsf{New}(Y_0, AKey) \tag{13}$$

$AUTH_{kas}^{client}$ $$\mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.\hat{W}, k_{Y,Z}^{t \to k}) \supset \hat{Y} = \hat{T} \wedge \hat{Z} = \hat{K} \wedge \hat{W} = \hat{C} \tag{14}$$

$(9, -2, -1)$ $$\hat{X_0} = \hat{T} \wedge \hat{K_0} = \hat{K} \wedge \hat{C_0} = \hat{C} \tag{15}$$

$(7, -1)$ $$[\mathbf{Client}]_C\ \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k_{S,T}^{s \to t}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S})) \tag{16}$$

G Formal Description of Kerberos with PKINIT

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};
  $]_S$

H Proof of PKINIT Security Properties

H.1 Environmental Assumptions

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.

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

H.2 Additional Axioms

A1 $\mathsf{New}(X, n) \wedge \mathsf{New}(Y, n) \supset X = Y$

A2 $\mathsf{SymEnc}(X, m, k) \supset \mathsf{Possess}(X, k) \wedge \mathsf{Possess}(X, m)$

A3 $\mathsf{New}(X, k) \wedge HASH[k](m) = HASH[k](m') \supset m = m'$

H.3 Proofs of $AUTH^{client}_{kas}$

$$[\mathbf{Client}]_C\ \mathsf{Verify}(C, SIG[sk_K](k.ck), vk_K) \tag{1}$$

(-1) $$[\mathbf{Client}]_C\ \exists \eta.\ \mathsf{Sign}((\hat{K}, \eta), k.ck, sk_K) \tag{2}$$

Inst $(\hat{K}, \eta) \mapsto K$ $$[\mathbf{Client}]_C\ \mathsf{Sign}(K, k.ck, sk_K) \tag{3}$$

HON $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Sign}(X, n.m, sk_X) \wedge \neg\mathsf{Time}(n) \supset \mathsf{New}(X, n) \wedge \exists m'.\ m = HASH[k](m') \wedge \mathsf{Hash}(X, m', k) \tag{4}$$

$(-2, -1)$ $$[\mathbf{Client}]_C\ \mathsf{New}(K, k) \wedge \exists m'.\ ck = HASH[k](m') \wedge \mathsf{Hash}(K, m', k) \tag{5}$$

$$[\mathbf{Client}]_C\ ck = HASH[k](Cert_C.sigc.\hat{C}.\hat{T}.n_1) \tag{6}$$

A3, $(-2, -1)$ $$[\mathbf{Client}]_C\ m' = Cert_C.sigc.\hat{C}.\hat{T}.n_1 \tag{7}$$

$(-3, -1)$ $$[\mathbf{Client}]_C\ \mathsf{New}(K, k) \wedge \mathsf{Hash}(K, Cert_C.sigc.\hat{C}.\hat{T}.n_1, k) \tag{8}$$

HON $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Hash}(X, Cert_{C_0}.sigc_0.\hat{C}_0.\hat{T}_0.n, k_0) \supset \exists t, Key.\ \mathsf{Send}(X, E_{pk}[pk_{C_0}](Cert_X.SIG[sk_X](k_0.\ HASH[k_0](Cert_{C_0}.sigc_0.\hat{C}_0.\hat{T}_0.n)).\hat{C}_0.E_{sym}[k^{t\rightarrow k}_{T_0,X}](Key.\hat{C}_0).\ E_{sym}[k_0](Key.n.t.\hat{T}_0))) \tag{9}$$

$(-2, -1)$ $$[\mathbf{Client}]_C\ \exists t, Key.\ \mathsf{Send}(K, E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)).\hat{C}.\ E_{sym}[k^{t\rightarrow k}_{T,K}](Key.\hat{C}).E_{sym}[k](Key.n_1.t.\hat{T})) \tag{10}$$

The last equation establishes $A\tilde{U}TH^{kas}_{client}$: $[\mathbf{Client}]_C\ \exists t_K, AKey.\ AUTH_{kas}$. In section H.5, We will use $A\tilde{U}TH^{kas}_{client}$ to prove $[\mathbf{Client}]_C\ \mathsf{GoodProtocol}(k, \{dk_C\})$ and $\mathsf{GoodInit}(k, \{dk_C\}) \equiv \mathsf{Hon}(\hat{C}, \hat{K})$. Therefore, using axiom **CTXL** we can derive:

$$\mathbf{CTXL} \quad [\mathbf{Client}]_C\ \mathsf{SymDec}(X, E_{sym}[k](m), k) \Rightarrow \exists Z.\ \mathsf{SymEnc}(Z, m, k) \tag{11}$$

Now we proceed to prove $AUTH^{kas}_{client}$:

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[k](AKey.n_1.t_K.\hat{T}), k) \tag{12}$$

$$(11, -1) \quad [\mathbf{Client}]_C\ \exists Z_0.\ \mathsf{SymEnc}(Z_0, AKey.n_1.t_K.\hat{T}, k) \tag{13}$$

$$\text{Inst } Z_0 \mapsto Z, (-1) \quad [\mathbf{Client}]_C\ \mathsf{SymEnc}(Z, AKey.n_1.t_K.\hat{T}, k) \tag{14}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.n.t.\hat{T}_0, k_0) \supset \mathsf{New}(X, k_0) \tag{15}$$

$$(-2, -1) \quad [\mathbf{Client}]_C\ \mathsf{New}(Z, k) \tag{16}$$

$$\mathbf{A1}, (-1) \quad [\mathbf{Client}]_C\ Z = K \tag{17}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{Hash}(X, Cert_{C_0}.sigc_0.\hat{C}_0.\hat{T}_0.n, k_0) \supset \exists t, Key.\ \mathsf{SymEnc}(X, Key.n.t.\hat{T}_0, k_0) \tag{18}$$

$$(-1) \quad [\mathbf{Client}]_C\ \exists t, Key.\ \mathsf{SymEnc}(K, Key.n_1.t.\hat{T}, k) \tag{19}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.n.t.\hat{T}_0, k_0) \wedge \mathsf{SymEnc}(X, Key'.n'.t'.\hat{T}'_0, k_0) \supset Key = Key' \wedge n = n' \wedge t = t' \wedge \hat{T}_0 = \hat{T}'_0 \tag{20}$$

$$A\tilde{U}TH^{client}_{kas}, (-1) \quad [\mathbf{Client}]_C\ \mathsf{Send}(K, E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)).\hat{C}.\ E_{sym}[k^{t\to k}_{T,K}](Key.\hat{C}).E_{sym}[k](AKey.n_1.t_K.\hat{T})) \tag{21}$$

$$(-1) \quad AUTH^{client}_{kas} \tag{22}$$

H.4 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 $E_{sym}[k^{type}_{X,Y}](M)^r$. 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.

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

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

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

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

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

$$\mathsf{Hon}(\hat{X},\hat{Y}),\Gamma_0,(-1) \quad [\mathbf{Role}]_X\ \hat{Z_0} = \hat{X} \vee \hat{Z_0} = \hat{Y} \tag{6}$$

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

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

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

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

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

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

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

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

$$[\mathbf{TGS}]_T\ \exists \eta.\ \mathsf{SymEnc}((\hat{K},\eta), (AKey, \hat{C}), k^{t\to k}_{T,K}) \tag{12}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C_0}, k^{t\to k}_{Y,X}) \supset \exists k_0, ck_0, n, t.\ \mathsf{Send}(X, E_{pk}[pk_{C_0}](Cert_X.SIG[sk_X](k_0.ck_0)).\ \hat{C_0}.E_{sym}[k^{t\to k}_{Y,X}](Key.\hat{C_0}).E_{sym}[k_0](Key.n.t.\hat{Y})) \tag{13}$$

$$\mathsf{Hon}(\hat{K}),(-2,-1) \quad [\mathbf{TGS}]_T\ \exists \eta, n_1, k, ck, t_K.\ \mathsf{Send}((\hat{K},\eta), E_{pk}[pk_C](Cert_K.SIG[sk_K](k.ck)).\ \hat{C}.E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C}).E_{sym}[k](AKey.n_1.t_K.\hat{T})) \tag{14}$$

$$(-1) \quad AUTH^{tgs}_{kas} \tag{15}$$

Instantiating for $AUTH_{tgs}^{server}$:

$$[\mathbf{Server}]_S \; \exists \eta.\, \mathsf{SymEnc}((\hat{T}, \eta), SKey.\hat{C}, k_{S,T}^{s \to t}) \tag{16}$$

$$\mathbf{HON} \quad \mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{C}_0, k_{Y,X}^{s \to t}) \supset \exists n, Key'.\, \mathsf{Send}(X, \hat{C}_0.E_{sym}[k_{Y,X}^{s \to t}](Key.\hat{C}_0).E_{sym}[Key'](Key.n.\hat{Y})) \tag{17}$$

$$\mathsf{Hon}(\hat{T}), (-2,-1) \quad [\mathbf{Server}]_S \; \exists \eta, n, Key'.\, \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k_{S,T}^{s \to t}](SKey.\hat{C}).E_{sym}[Key'](SKey.n.\hat{S})) \tag{18}$$

$$(-1) \quad AUTH_{tgs}^{server} \tag{19}$$

H.5 Proofs of $SEC_{k}^{client}, SEC_{k}^{kas}$

The assumed condition Φ is the conjunction of the following formulae:

$\Phi_0 : \forall X.\, \mathsf{New}(X, k) \supset \hat{X} = \hat{K}$

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

$\Phi_2 : \forall X, t.\, \mathsf{New}(X, k) \supset \neg\mathsf{Sign}(X, t.k, sk_X)$

$\Phi_3 : \forall X, Y, ck_0.\, \mathsf{New}(X, k) \wedge \mathsf{PkEnc}(X, Cert_X.SIG[sk_X](k.ck_0), pk_Y) \supset \hat{Y} = \hat{C}$

$\Phi_4 : \forall X, m, key.\, \mathsf{New}(X, k) \supset \neg\mathsf{SymEnc}(X, k.m, key)$

H.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:

$\Phi_0 : \forall X.\, \mathsf{New}(X, AKey) \supset \hat{X} = \hat{K}$

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

$\Phi_2 : \forall X, m.\, \mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.m, k_0) \supset k_0 \in \{k, k_{T,K}^{t \to k}\}$

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

$\Phi_4 : \forall X, t.\, \mathsf{New}(X, AKey) \supset \neg(\mathsf{Sign}(X, M, sk_X) \wedge \mathsf{ContainsOpen}(M, AKey))$

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

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

Now we present the formal proof of goodness:

$$\begin{aligned}
\text{Let, } [\mathbf{Client}_1]_{C'} : [&\texttt{new } n'_1; \texttt{ new } \tilde{n'}_1;\\
&sigc' := \texttt{sign } t'_C.\tilde{n'}_1, sk_{C'};\\
&\texttt{send } Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n'_1;]_{C'}
\end{aligned}$$

$$[\mathbf{Client}_1]_{C'}\ \mathsf{New}(C', n'_1) \wedge \mathsf{Send}(C', \hat{C'}.\hat{T'}.n'_1) \wedge \mathsf{Sign}(C', t'_C.\tilde{n'}_1, sk_{C'}) \tag{1}$$

$$\Phi_1, \Phi_4, (-1) \quad [\mathbf{Client}_1]_{C'}\ n'_1 \neq AKey \wedge \tilde{n'}_1 \neq AKey \tag{2}$$

$$\mathbf{G1}, \mathbf{G5}, \mathbf{G11}, (-1) \quad [\mathbf{Client}_1]_{C'}\ \mathsf{Good}(C', Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n'_1, AKey, \mathcal{K}) \tag{3}$$

$$\mathbf{SG1\text{-}2}, (-1) \quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_1]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{4}$$

$$\begin{aligned}
\text{Let, } [\mathbf{Client}_2]_{C'} : [&\texttt{receive } encp'_{kc}.\hat{C'}.tgt'.enc'_{kc};\\
&textp'_{kc} := \texttt{pkdec } encp'_{kc}, dk_{C'};\\
&\texttt{match } textp'_{kc} \texttt{ as } Cert_{K'}.sigk';\\
&\texttt{verify } sigk', k'.ck', vk'_K;\\
&\tilde{ck'} := \texttt{hash } Cert_{C'}.sigc'.\hat{C'}.\hat{T'}.n'_1, k';\\
&\texttt{match } \tilde{ck'} \texttt{ as } ck';\\
&text'_{kc} := \texttt{symdec } enc'_{kc}, k';\\
&\texttt{match } text'_{kc} \texttt{ as } AKey'.n'_1.t'_K.\hat{T'};]_{C'}
\end{aligned}$$

$$\mathbf{SG1} \quad \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_2]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{5}$$

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

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

$$[\mathbf{Client}_3]_{C'}\ \mathsf{New}(C', n'_2) \wedge \mathsf{Send}(C', tgt'.enc'_{ct}.\hat{C'}.\hat{S'}.n'_2) \tag{6}$$

$$\Phi_1, (-1) \quad [\mathbf{Client}_3]_{C'}\ n'_2 \neq AKey \tag{7}$$

$$\mathbf{G1}, (-1) \quad \theta_3\ [\texttt{new } n'_2;]_{C'}\ \mathsf{Good}(C', tgt', AKey, \mathcal{K}) \wedge \mathsf{Good}(C', n'_2, AKey, \mathcal{K}) \tag{8}$$

$$\mathbf{G}*, (-1) \quad \theta_3\ [\mathbf{Client}_3]_{C'}\ \mathsf{Good}(C', tgt'.enc'_{ct}.\hat{C'}.\hat{S'}.n'_2, AKey, \mathcal{K}) \tag{9}$$

$$\mathbf{SG1\text{-}2}, (-1) \quad \theta_3 \wedge \mathsf{SendGood}(C', AKey, \mathcal{K})\ [\mathbf{Client}_3]_{C'}\ \mathsf{SendGood}(C', AKey, \mathcal{K}) \tag{10}$$

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

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

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

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (11)

Precondition $\theta_5 : \mathsf{Good}(C', st', AKey, \mathcal{K})$

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

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

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

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

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (12)

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

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

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

$\mathsf{SendGood}(C', AKey, \mathcal{K})$ (13)

Let, $[\mathbf{KAS}]_{K'}$: [$\texttt{receive}\ Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1$;

$\texttt{verify}\ sigc', t'_C.\tilde{n'_1}, vk_{C'}$;

$\texttt{new}\ k'; \texttt{new}\ AKey'$;

$ck' := \texttt{hash}\ Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1, k'$;

$sigk' := \texttt{sign}\ k'.ck', sk_{K'}$;

$encp'_{kc} := \texttt{pkenc}\ Cert_{K'}.sigk', pk_{C'}$;

$tgt' := \texttt{symenc}\ AKey'.\hat{C}', k^{t\rightarrow k}_{T',K'}$;

$enc'_{kc} := \texttt{symenc}\ AKey'.n'_1.t_{K'}.\hat{T}', k'$;

$\texttt{send}\ encp'_{kc}.\hat{C}'.tgt'.enc'_{kc};]_{K'}$

$[\mathbf{KAS}]_{K'}\ \mathsf{Sign}(K', k'.ck', sk_{K'})$ (14)

$\Phi_4, (-1)$ $[\mathbf{KAS}]_{K'}\ k' \neq AKey$ (15)

Case 1 : $AKey' = AKey$

$[\mathbf{KAS}]_{K'}\ \mathsf{New}(K', AKey) \wedge \mathsf{SymEnc}(K', AKey.\hat{C}', k^{t\rightarrow k}_{T',K'})$

$\wedge\ \mathsf{SymEnc}(K', AKey.n'_1.t_{K'}.\hat{T}', k')$ (16)

$\Phi_2, (-1)$ $[\mathbf{KAS}]_{K'}\ k' \in \{k, k_{T,K}^{t \to k}\} \wedge k_{T',K'}^{t \to k} \in \{k, k_{T,K}^{t \to k}\}$ (17)

(-1) $[\mathbf{KAS}]_{K'}\ k' \in \mathcal{K} \wedge k_{T',K'}^{t \to k} \in \mathcal{K}$ (18)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(K', encp'_{kc}.\hat{C}'.tgt'.enc'_{kc}, AKey, \mathcal{K})$ (19)

Case 2 : $AKey' \neq AKey$

G2 $[\texttt{receive}\ Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1;]_{K'}\ \mathsf{Good}(K', Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1, AKey, \mathcal{K})$ (20)

$\mathbf{G6}, (-1)$ $[\texttt{receive}\ Cert_{C'}.sigc'.\hat{C}'.\hat{T}'.n'_1;]_{K'}\ \mathsf{Good}(K', n'_1, AKey, \mathcal{K})$ (21)

$\mathbf{G}*, (-1)$ $[\mathbf{KAS}]_{K'}\ \mathsf{Good}(encp'_{kc}.\hat{C}'.tgt'.enc'_{kc}, AKey, \mathcal{K})$ (22)

$\mathbf{SG1\text{-}2}, (-4, -1)$ $\mathsf{SendGood}(K', AKey, \mathcal{K})\ [\mathbf{KAS}]_{K'}\ \mathsf{SendGood}(K', AKey, \mathcal{K})$ (23)

Let, $[\mathbf{TGS}]_{T'}$: $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;$
$text'_{ct1} := \texttt{symdec}\ enc'_{ct1}, k_{T',K'}^{t \to k};$
$\texttt{match}\ text'_{ct1}\ \texttt{as}\ AKey'.\hat{C}';$
$text'_{ct2} := \texttt{symdec}\ enc'_{ct2}, AKey';$
$\texttt{match}\ text'_{ct2}\ \texttt{as}\ \hat{C}';$
$\texttt{new}\ SKey';$
$st' := \texttt{symenc}\ SKey'.\hat{C}', k_{S',T'}^{s \to t};$
$enc'_{tc} := \texttt{symenc}\ SKey'.n'_2.\hat{S}', AKey';$
$\texttt{send}\ \hat{C}'.st'.enc'_{tc};]_{T'}$

G2, G6 $[\texttt{receive}\ enc'_{ct1}.enc'_{ct2}.\hat{C}'.\hat{S}'.n'_2;]_{T'}\ \mathsf{Good}(T', n'_2, AKey, \mathcal{K})$ (24)

$[\mathbf{TGS}]_{T'}\ \mathsf{New}(T', SKey') \wedge \mathsf{SymEnc}(T', SKey'.\hat{C}', k_{S',T'}^{s \to t})$ (25)

$\Phi_3, (-1)$ $[\mathbf{TGS}]_{T'}\ SKey' \neq AKey$ (26)

$\mathbf{G1}, (-1)$ $[\cdots; \texttt{new}\ SKey';]_{T'}\ \mathsf{Good}(T', SKey', AKey, \mathcal{K})$ (27)

$\mathbf{G}*, (-4, -1)$ $[\mathbf{TGS}]_{T'}\ \mathsf{Good}(T', \hat{C}'.st'.enc'_{tc}, AKey, \mathcal{K})$ (28)

$\mathbf{SG1\text{-}2}, (-1)$ $\mathsf{SendGood}(T', AKey, \mathcal{K})\ [\mathbf{TGS}]_{T'}\ \mathsf{SendGood}(T', AKey, \mathcal{K})$ (29)

Let, $[\mathbf{Server}]_{S'}$: [`receive` $enc'_{cs1}.enc'_{cs2}$;
$text'_{cs1} :=$ `symdec` $enc'_{cs1}, k^{s\rightarrow t}_{S',T'}$; `match` $text'_{cs1}$ `as` $SKey'.\hat{C}'$;
$text'_{cs2} :=$ `symdec` $enc'_{cs2}, SKey'$; `match` enc'_{cs2} `as` $\hat{C}'.t'$;
$enc'_{sc} :=$ `symenc` $t', SKey'$;
`send` enc'_{sc}; $]_{S'}$

Case 1: $SKey' \in \mathcal{K}$

G7, (-1) $[\cdots; enc'_{sc} :=$ `symenc` $t', SKey';]_{S'}$ $\mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (30)

Case 2: $SKey' \notin \mathcal{K}$ (31)

G2, **G6** [`receive` $enc'_{cs1}.enc'_{cs2}$; $]_{S'}$ $\mathsf{Good}(S', enc'_{cs2}, AKey, \mathcal{K})$ (32)

G6, **G8**, (-1) $[\cdots; text'_{cs2} :=$ `symdec` $enc'_{cs2}, SKey'$; `match` enc'_{cs2} `as` $\hat{C}'.t';]_{S'}$
$\mathsf{Good}(S', t', AKey, \mathcal{K})$ (33)

G7, (-1) $[\cdots; enc'_{sc} :=$ `symenc` $t', SKey';]_{S'}$ $\mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (34)

$(-4, -1)$ $[\mathbf{Server}]_{S'}$ $\mathsf{Good}(S', enc'_{sc}, AKey, \mathcal{K})$ (35)

SG1-2, (-1) $\mathsf{SendGood}(S', AKey, \mathcal{K})$ $[\mathbf{Server}]_{S'}$ $\mathsf{SendGood}(S', AKey, \mathcal{K})$ (36)

Theorem 10 $\Phi \supset \mathsf{GoodProtocol}(AKey, \mathcal{K})$ (37)

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:

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

Combining everything we have:

$$PKINIT \vdash SEC^{client}_{akey}, SEC^{kas}_{akey}, SEC^{tgs}_{akey}$$

H.7 Proofs of $SEC^{client}_{skey}, SEC^{tgs}_{skey}$

In this section we formally prove the secrecy of the session key $SKey$ with respect to the set of keys $\mathcal{K} = \{AKey, k^{s\rightarrow t}_{S,T}\}$. The assumed condition Φ is the conjunction of the following formulae:

$\Phi_0 : \forall X.\, \mathsf{New}(X, SKey) \supset \hat{X} = \hat{T}$

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

$\Phi_2 : \forall X, m.\, \mathsf{New}(X, SKey) \wedge \mathsf{SymEnc}(X, SKey.m, k_0) \supset k_0 \in \{AKey, k^{s\rightarrow t}_{S,T}\}$

$\Phi_3 : \forall X, \hat{T_0}, \hat{C_0}.\, \mathsf{New}(X, SKey) \supset \neg\mathsf{SymEnc}(X, SKey.\hat{C_0}, k^{t\rightarrow k}_{T_0,X})$

$\Phi_4 : \forall X, t.\, \mathsf{New}(X, SKey) \supset \neg(\mathsf{Sign}(X, M, sk_X) \wedge \mathsf{ContainsOpen}(M, SKey))$

H.8 Proof of $AUTH_{tgs}^{client}$

We have derived in Appendix H.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:

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(Z, E_{sym}[AKey](m), AKey) \Rightarrow \exists X.\ \mathsf{SymEnc}(X, m, AKey) \quad (1)$$

Now, the thread $[\mathbf{Client}]_C$ decrypts the message $E_{sym}[AKey](SKey.n_2.\hat{S})$. Hence we proceed:

$$[\mathbf{Client}]_C\ \mathsf{SymDec}(C, E_{sym}[AKey](SKey.n_2.\hat{S}), AKey) \quad (2)$$

$(-1, 1)$ $$[\mathbf{Client}]_C\ \exists X.\ \mathsf{SymEnc}(X, SKey.n_2.\hat{S}, AKey) \quad (3)$$

Inst $X \mapsto X_0, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{SymEnc}(X_0, SKey.n_2.\hat{S}, AKey) \quad (4)$$

(-1) $$[\mathbf{Client}]_C\ \mathsf{Possess}(X_0, AKey) \quad (5)$$

$SEC_{AKey}^{client}, (-1)$ $$\hat{X_0} = \hat{C} \wedge \hat{X_0} = \hat{K} \wedge \hat{X_0} = \hat{T} \quad (6)$$

Inst , $AUTH_{kas}^{client}$ $$[\mathbf{Client}]_C\ \mathsf{New}(K, AKey) \wedge \forall m.\ \neg\mathsf{SymEnc}(K, m, AKey) \quad (7)$$

$(-4, -1)$ $$[\mathbf{Client}]_C\ \neg\mathsf{New}(X_0, AKey) \quad (8)$$

HON
$$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key'.n.\hat{S_0}, Key) \wedge \neg\mathsf{New}(X, Key) \supset \exists \hat{K_0}, \hat{C_0}.\ \mathsf{SymDec}(X, E_{sym}[k_{X,K_0}^{t \to k}](Key.\hat{C_0}), k_{X,K_0}^{t \to k}) \wedge \mathsf{Send}(X, \hat{C_0}.E_{sym}[k_{S_0,X}^{s \to t}](Key'.\hat{C_0}).E_{sym}[Key](Key'.n.\hat{S_0})) \quad (9)$$

Inst, $(-4, -1)$
$$[\mathbf{Client}]_C\ \mathsf{SymDec}(X_0, E_{sym}[k_{X_0,K_0}^{t \to k}](AKey.\hat{C_0}), k_{X_0,K_0}^{t \to k}) \wedge \mathsf{Send}(X_0, \hat{C_0}.E_{sym}[k_{S,X_0}^{s \to t}](SKey.\hat{C_0}).E_{sym}[AKey](SKey.n_2.\hat{S})) \quad (10)$$

CTX0, (-1) $$[\mathbf{Client}]_C\ \exists Y.\ \mathsf{SymEnc}(Y, AKey.\hat{C_0}, k_{X_0,K_0}^{t \to k}) \quad (11)$$

Inst $Y \mapsto Y_0, (-1)$ $$[\mathbf{Client}]_C\ \mathsf{SymEnc}(Y_0, AKey.\hat{C_0}, k_{X_0,K_0}^{t \to k}) \quad (12)$$

HON $$\mathsf{Honest}(\hat{X}) \wedge \mathsf{SymEnc}(X, Key.\hat{W}, k_{X,Z}^{t \to k}) \supset \mathsf{New}(X, Key) \quad (13)$$

$(-4, -1)$ $$[\mathbf{Client}]_C\ \mathsf{New}(Y_0, AKey) \quad (14)$$

$AUTH_{kas}^{client}$
$$\mathsf{New}(X, AKey) \wedge \mathsf{SymEnc}(X, AKey.\hat{W}, k_{Y,Z}^{t \to k}) \supset \hat{Y} = \hat{T} \wedge \hat{Z} = \hat{K} \wedge \hat{W} = \hat{C} \quad (15)$$

$(10, -2, -1)$ $$\hat{X_0} = \hat{T} \wedge \hat{K_0} = \hat{K} \wedge \hat{C_0} = \hat{C} \quad (16)$$

$(10, -1)$ $$[\mathbf{Client}]_C\ \exists \eta.\ \mathsf{Send}((\hat{T}, \eta), \hat{C}.E_{sym}[k_{S,T}^{s \to t}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S})) \quad (17)$$