A Reduction-Based Proof for Authentication and Session Key Security in 3-Party Kerberos

Jörg Schwenk[1] and Douglas Stebila[2]

[1]Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany, joerg.schwenk@rub.de
[2]University of Waterloo, Canada, dstebila@uwaterloo.ca

July 3, 2019

Abstract

Kerberos is one of the earliest network security protocols, providing authentication between clients and servers with the assistance of trusted servers. It remains widely used, notably as the default authentication protocol in Microsoft Active Directory (thus shipped with every major operating system), and is the ancestor of modern single sign-on protocols like OAuth and OpenID Connect.

There have been many analyses of Kerberos in the symbolic (Dolev–Yao) model, which is more amenable to computer-aided verification tools than the computational model, but also idealizes messages and cryptographic primitives more. Reduction-based proofs in the computational model can provide assurance against a richer class of adversaries, and proofs with concrete probability analyses help in picking security parameters, but Kerberos has had no such analyses to date.

We give a reduction-based security proof of Kerberos authentication and key establishment, focusing on the mandatory 3-party mode. We show that it is a secure authentication protocol under standard assumptions on its encryption scheme; our results can be lifted to apply to quantum adversaries as well.

As has been the case for other real-world authenticated key exchange (AKE) protocols, the standard AKE security notion of session key indistinguishability cannot be proven for Kerberos since the session key is used in the protocol itself, breaking indistinguishability. We provide two positive results despite this: we show that the standardized but optional sub-session mode of Kerberos does yield secure session keys, and that the hash of the main session key is also a secure session key under Krawczyk's generalization of the authenticated and confidential channel establishment (ACCE) model.

1 Introduction

Kerberos was developed by Miller and Neuman at MIT to protect network services in a distributed computing project Athena. Initially, Kerberos exclusively used symmetric key primitives and was based on the Needham–Schroeder protocol [NS78a]. Early versions were internal to MIT; version 4 was published in [SNS88], and version 5 was standardized by the Internet Engineering Task Force [KN93, NYHR05].

The Kerberos protocol allows a client to authenticate to a server with the help of trusted server(s). The client and server each share long-term keys with a *Kerberos Authentication Server (KAS)*, but not with each other. Kerberos can be run in two modes: a two-exchange/three-party mode, or a three-exchange/four-party mode.

In the two-exchange/three-party case (which is mandatory-to-implement in the standard and the focus of this paper), the initial exchange, called the *Authentication Service (AS) exchange* [NYHR05, §3.1], takes place between the client and the KAS. The KAS provides the client a credential, part of which it decrypts and part of which it relays to the final server. The last exchange is called the *Client/Server Authentication (CS) exchange* [NYHR05, §3.2], in which the client and server authenticate each other using a challenge-response protocol.

In the three-exchange/four-party case (which is optional to implement), a middle exchange, called the *Ticket Granting Service (TGS) exchange* [NYHR05, §3.3] takes place between the client and the *Ticket Granting Server*. Here, the client presents its credential from the KAS (called a *Ticket Granting Ticket (TGT)*) to the TGS, which then issues a ticket for the final server which the client uses in the Client/Server Authentication exchange mentioned above. The main form of Kerberos described above relies solely on symmetric key primitives. There is a public-key variant of Kerberos [ZT06] which we do not consider in this paper.

Kerberos does not explicitly define a secure channel which follows session establishment. An optional feature allows transmission of a "sub-session key". Two separate RFCs [Rae05b, ZJH05] describe mechanisms by which the Kerberos session key can be used for encryption specifically or more generally exported to other cryptographic applications.

Adoption. Version 5 of Kerberos eliminated some security problems with the initial protocol, and was subsequently adopted in Microsoft Windows 2000 and all later versions as the default authentication protocol in Microsoft Active Directory. Many other operating systems support Kerberos, including FreeBSD, Linux, and macOS. In large Microsoft environments running Active Directory, Kerberos is used to implement access controls to all critical systems. It is thus a target for *advanced persistent threat (APT)* attackers: when Kerberos is run in three-exchange/four-party case, a ticket granting ticket acts as a "golden ticket" that can be used to gain access to services anywhere on the network [Met15]. Kerberos is also the progenitor of many single sign-on protocols where the same communication pattern is used, including Microsoft Passport, OAuth [Har12], OpenID, OpenID Connect, and SAML. Kerberos is constantly being updated to meet new security requirements, with 6 RFCs published since 2016.

1.1 Related work

Three-party communication protocols. There is a large class of three-party protocols, in which two parties C and S want to communicate with the assistance of a trusted party TP. In symmetric-key three-party protocols, C and S each share a symmetric key with TP, and the goal of the protocol is to perform authentication between C and S, and possibly establish a secret key or a secure channel. These protocols reduce the effort for key setup and key management: instead of initializing $O(n^2)$ keys between every pair of n parties, we only need to initialize n keys between each party and the trusted third party.

Figure 1 shows the three common communication patterns for three-party protocols as identified by Schwenk [Sch16]. Two-exchange/three-party Kerberos is an example of pattern 1, where the client C relays any messages between TP and the server S; other examples of pattern 1 include Needham–Schroeder [NS78b] as well as single sign-on protocols like OAuth [Har12]. In pattern 2, the server S relays communication between the client and TP; examples include Otway–Rees [OR87] and the Extended Authentication Protocol (EAP) [ABV$^+$04]. In pattern 3, TP mediates all

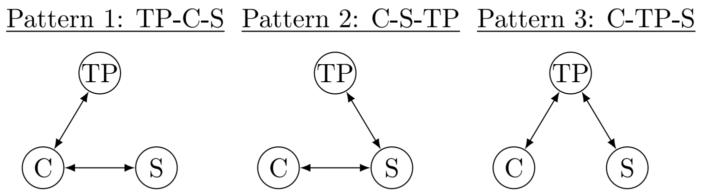

Figure 1: Communication patterns for 3-party protocols between a client C, server S, and trusted party TP, based on [Sch16, Fig. 1].

communication between C and S; examples include the "wide-mouthed frog" protocol [BAN90] and [BR95].

Symbolic modelling versus computational modelling. There are two main approaches to modelling security of protocols. In the *symbolic approach*, known as the *Dolev–Yao model*, messages and cryptographic operations are abstracted into an algebra on symbolic values, and proofs are based on logical or algebraic reasoning. In the *computational approach* (sometimes called "provable security"), messages remain as bitstrings and cryptographic operations are algorithms applied to bitstrings, and proofs are based on probabilistic arguments and complexity-theoretic reductions to (well-established) cryptographic assumptions like factoring or the Diffie–Hellman problem. Automated tools for proof generation and verification are more readily applied in the symbolic model and reduce the risk of errors in manual proofs, whereas the computational model can often admit richer adversaries, and reductionist proofs with concrete probability analyses can help in picking security parameters.

Three-party protocols, including Kerberos, have been extensively studied in the symbolic model. A full survey of this literature is beyond the scope of this paper, but we highlight a few works. The seminal paper of Burrows, Abadi, and Needham [BAN90] developed a symbolic logic for analysis of authentication protocols and initiated the study of three-party protocols in this setting, including Needham–Schroeder and Kerberos, from which has followed a long line of symbolic analyses of both Needham–Schroeder [Low96, Mea96, Bac04, BP03, WGC12, SB12] and Kerberos [BR97, BP98b, BCJ+06b, KEGM14]; we will further discuss some of the results on Kerberos below.

Reduction-based proofs are extensively used in the literature for all types of cryptographic primitives and protocols. For our purposes, the most relevant are of authentication and authenticated key exchange protocols, of which the seminal work is [BR94]. Most deal with either (a) two-party protocols, in which each party has a different role, i.e., client/server; or (b) group key agreement protocols [BD95, STW96, STW00], in which each party has the same role, i.e., they all execute the same code. For three-party protocols, Bellare and Rogaway [BR95] gave the first security model analysis for three-party key distribution, focusing on communication pattern 3 of Figure 1. Brzuska and Jacobsen [BJ17] gave a reduction-based proof for the 3-party Extensible Authentication Protocol (EAP), which follows pattern 2 in Figure 1; EAP differs from Kerberos in its communication pattern and that it involves public keys. Bhargavan et al [BBF+17] describe a three-party secure channel notion (3ACCE) and use it to analyze a proxied form of TLS.

There have also been serious efforts to unify the symbolic and computational models by developing methodologies to translate results in the Dolev–Yao model to the computational model, often provided certain requirements are met. Several works have applied this to Kerberos, which we discuss below.

1.2 Related work on Kerberos

Kerberos version 4 was first presented to the academic community in 1988 [SNS88] and version 5 in 1989 [Koh90]. Kerberos has been subject to many modifications, integrations, and performance studies, which we omit from this section.

A variety of weaknesses and limitations have been identified in Kerberos. Bellovin and Merritt [BM91] discussed a number of limitations which still exist, e.g. the vulnerability to replay attacks within the ticket's lifetime, and the usage of insecure time. A weakness was discovered in a weak random number generator algorithm in Kerberos version 4 [DLS97], and [YHR04] showed the dangers of using unauthenticated encryption: they were able to forge arbitrary tickets by modifying the ciphertext.

Starting in 1995, public key versions of Kerberos have been proposed [McM95, Gan95, SC97, ZT06]. A man-in-the-middle attack on one public key version was reported in [CJS$^+$08], and mitigations have been proposed and implemented.

Symbolic modelling of Kerberos. After Burrows, Abadi, and Needham [BAN90], Bella et al. in a series of papers [BR97, BP98b, BP98a], initiated the Dolev–Yao style analysis of Kerberos. This work was continued later by Butler et al. [BCJS02, BCJS03, BCJ$^+$06b]. Using different tools like CSP, BAN logic and strand spaces, this work was continued in [LYZZ09, FLW09, AHSM10, LP10]. Sprenger and Basin [SB12] use a theorem prover to analyze several AKE protocols, including Kerberos, in the Dolev–Yao model.

Computational modelling of Kerberos. Boldyreva and Kumar [BK11] investigated the different encryption modes used in Kerberos. They showed that a modified form of Kerberos version 5's general encryption profile [Rae05b, §6] and the simplified profile [Rae05b, §5],[Rae05a] both satisfy standard definitions of authenticated encryption ($\mathsf{IND\text{-}CCA}$ and $\mathsf{INT\text{-}CTXT}$) under standard assumptions on the underlying cryptographic algorithms.

Turning to the authentication and key establishment portions of Kerberos, we find the works most closely related to ours. Two lines of work give proofs in a symbolic model and then use some methodology to translate to a computational model.

Backes et al. [BCJ$^+$06a, BCJ$^+$11] show in the BPW [BPW03] form of the Dolev–Yao model that each of Kerberos's three (or four) exchanges satisfies an authentication property as well as key indistinguishability for the sub-session key. The framework of Backes, Pfitzmann, and Waidner [BPW03] then lifts these results to the computational setting. [BCJ$^+$06a, BCJ$^+$11] also show results for the public key version.

Roy et al. [RDDM07] work in the computational version [DDM$^+$05] of the Protocol Composition Logic [DDMP03], which allows for abstract reasoning like in the symbolic model but translates results to the computational model, showing secrecy properties of the session key in Kerberos in two ways: (a) when Kerberos is truncated before the session key is used. They show that the session key is indistinguishable; (b) when Kerberos is not truncated, the session key can be safely used in an $\mathsf{IND\text{-}CCA}$-secure encryption scheme. A subset of the same authors shows results for the public key version [RDM07].

Blanchet et al. [BJST08] give computational proofs of Kerberos's three-exchange/four-party mode and public key mode using the CryptoVerif [Bla06] prover, specifically showing authentication properties separately for each of the three (or four) exchanges. They also show secrecy of the session key in truncated sessions, safe use of the session key from non-truncated sessions in $\mathsf{IND\text{-}CCA}$-secure encryption schemes, and indistinguishability of the sub-session key.

None of these models allow for adaptive corruptions of parties. Roy et al. do not allow for reveal of unrelated session keys. Both Backes et al. and Blanchet et al. [BJST08] consider the three-exchange/four-party mode, but both only analyze each exchange separately and provide no composition theorem to combine results on the three exchanges; this limits the adversary to attacking one exchange at a time, excluding attacks which might interleave components from different exchanges. Blanchet et al. provide a concrete analysis (showing quantitatively how the effort of breaking Kerberos relates to the effort of breaking the underlying primitives) for only one of their analyzed properties, and the other works above do not do this at all. Schwenk [Sch14] analyzes a modified version of Kerberos in which timestamps are replaced by nonces, and in which the session key is not used directly in the protocol; but [Sch14] does not address the security of the unmodified Kerberos protocol.

1.3 Contributions

In this paper, we give a reduction-based security proof for authentication and session key security of the two-exchange/three-party mode of Kerberos version 5; to the best of our knowledge, this work is the first reduction-based analysis which allows simultaneous sessions, and does not require modifications to the protocol.

Authentication. In Section 3, we develop a security definition for mutual authentication of clients and servers in 3-party protocols, where the third-party authentication server is stateless. Our definition is a Bellare–Rogaway-style model [BR94]. Our model is generic and can be applied to any such 3-party protocol, however the matching and freshness conditions are specialized to capture specific properties of Kerberos that are distinct from those in existing 3-party models such as [BR95]. The Kerberos-specific properties captured include:

- The authentication definition allows multiple servers to match a single client session, since Kerberos does not prevent replay attacks against servers.
- Matching in the authentication definition is based on session identifiers, not transcripts, since in Kerberos is no cryptographic binding between the c_1 ciphertext from the KAS to the client and the c_2 ciphertext that the client is supposed to relay from the KAS to the server.

These and other subtleties are discussed in Section 4.1.

(The 3ACCE model [BBF$^+$17] includes 4 security properties, one of which is entity authentication, which we could use to model authentication. However, 3ACCE would still not suffice for modelling the two forms of session key security we note below. As such, we go with a BR-style model rather than the 3ACCE framework so we can more readily model both authentication and session key security in the same framework.)

We proceed to prove in Section 5 that three-party Kerberos satisfies this authentication security definition, under the assumption that its symmetric encryption scheme is an authenticated encryption scheme ($\mathsf{IND\text{-}CPA}$ and $\mathsf{INT\text{-}CTXT}$) (an assumption justified by [BK11]). Our proof provides concrete probability bounds which can aid in selecting security parameters. Our proof is a game-hopping proof in the standard model, and each hop is either combinatorial, a guess, or involves a straight-line reduction. As a consequence, we can immediately apply Song's lifting lemma [Son14] to show that Kerberos is secure against quantum adversaries if its underlying primitives are. Given that basic Kerberos relies solely on symmetric primitives, and that public key cryptography is at the greatest risk from quantum computers, our results reinforce that Kerberos is a sound option for quantum resistance, assuming its underlying symmetric primitives are quantum-secure.[1]

Session key security. We next consider in Section 6 whether Kerberos establishes a secure session key. Unfortunately, the session key k_{CS} is used during the execution of the Kerberos protocol in the symmetric encryption scheme, so it no longer satisfies the standard indistinguishability security property required of authenticated key exchange (AKE) protocols. Recall that the adversary picks a "test" session and is given a value which is either the real key for that session or a random string; the adversary's challenge is to distinguish the two cases. Since the real session key is already used in the execution of the protocol, the adversary can check its challenge value by trial decrypting the protocol's ciphertexts. We follow two approaches to still prove useful results about Kerberos session key security.

First, we consider the Kerberos sub-session key. Kerberos permits the negotiation of an optional *sub-session key* via key transport from the client to the server or vice versa. We show that this key is indistinguishable from random in the typical sense from AKE models.

Second, we consider the use of the Kerberos session key in subsequent applications. The goal is to prove that the Kerberos session key remains suitable for use generically in other cryptographic schemes. The preferred reduction-based approach for doing so is the *authenticated and confidential channel establishment (ACCE)* which Jager et al. [JKSS12] introduced to resolve the problem of composing the TLS 1.2 handshake with its encryption layer. But the ACCE (and its 3-party variant 3ACCE [BBF$^+$17]) require a fully specified encryption layer in order to model channel security. RFC 3961 [Rae05b] does specify one way in which the Kerberos session key can be used in a subsequent encryption layer, but this is not the only allowed use of the Kerberos session key: for example, [ZJH05] describes a general mechanism by which the Kerberos session key can be exported, and this is still not exclusive. Thus there is no reason to believe it will be used outside Kerberos in the same encryption scheme inside Kerberos, and thus there is no specific encryption layer with which we could apply the ACCE or 3ACCE notions of channel security.

Instead, we turn to a generalization of the ACCE framework by Krawczyk [Kra16], which models the use of the session in an arbitrary subsequent protocol modelled by "functional queries" and tests the security of the session key when used in a "functional test". Ideally, we would show that the Kerberos session key is secure when used in any subsequent functional queries and functional test, but that is not possible: one could design degenerate schemes that lead to insecurity when used in combination with the symmetric encryption scheme used inside Kerberos. However, we can show that if the Kerberos session key is *hashed* before being passed to the application, then security is maintained. To do so, we construct a novel *hashed-key-hiding* property for symmetric encryption schemes, similar to the key-hiding property of Fischlin [Fis99]. We justify this assumption by showing it holds when the hash function is a random oracle and Kerberos' encryption scheme is secure against key recovery under chosen plaintext attacks.

This approach provides a more generic solution than works mentioned above [RDDM07, BJST08] which either truncated the key exchange to prove session indistinguishability, or only showed safe composition with an IND-CCA-secure encryption scheme, but without adaptive corruptions of parties and other model restrictions noted above.

[1]Symmetric key cryptography was thought, for a long time, to be relatively unperturbed by quantum algorithms, with the exception that one would have to double the key size due to Grover's algorithm. However, recent results [KLLN16] show that many authenticated encryption modes of operation are insecure when a quantum adversary has superposition access to the encryption oracle, but HMAC, which is used for authentication in Kerberos, is not affected by [KLLN16].

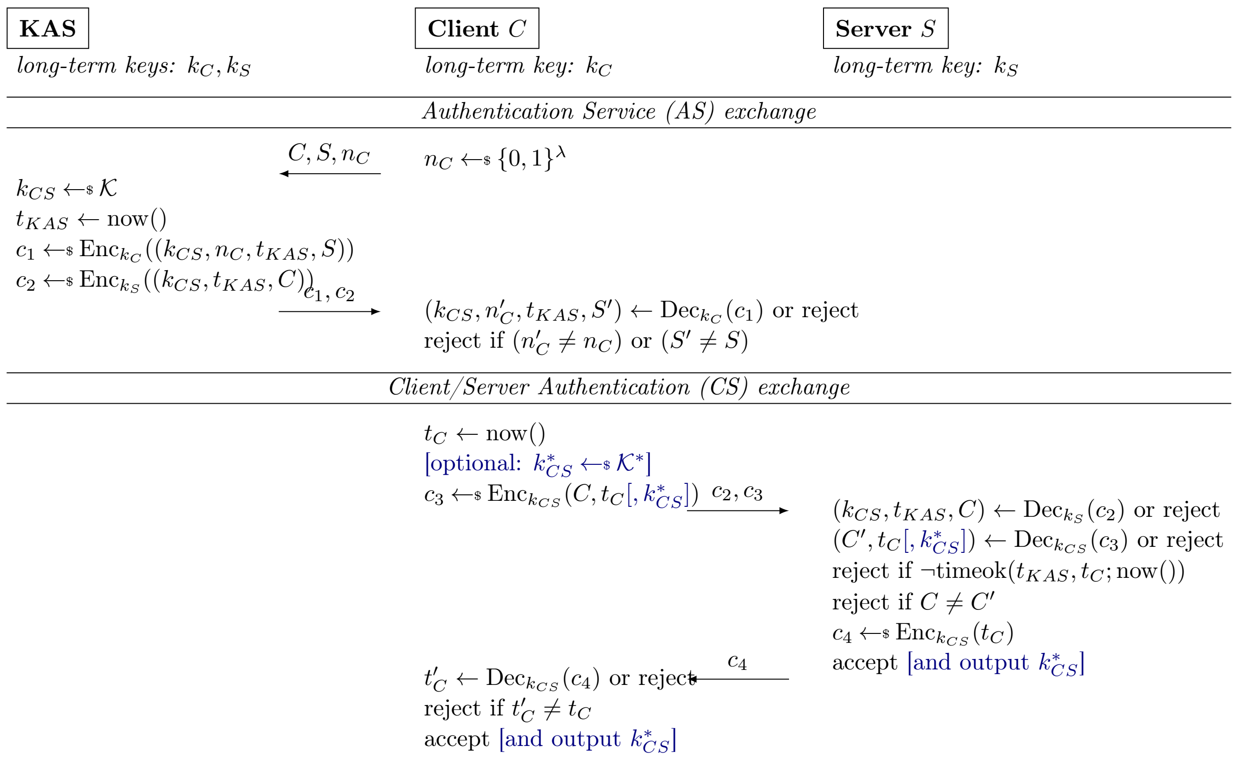
KAS
long-term keys: k_C, k_S
Client C
long-term key: k_C
Server S
long-term key: k_S
Authentication Service (AS) exchange
C, S, n_C
$n_C \leftarrow\$ \{0,1\}^\lambda$
$k_{CS} \leftarrow\$ \mathcal{K}$
$t_{KAS} \leftarrow \mathrm{now}()$
$c_1 \leftarrow\$ \mathrm{Enc}_{k_C}((k_{CS}, n_C, t_{KAS}, S))$
$c_2 \leftarrow\$ \mathrm{Enc}_{k_S}((k_{CS}, t_{KAS}, C))$
c_1, c_2
$(k_{CS}, n'_C, t_{KAS}, S') \leftarrow \mathrm{Dec}_{k_C}(c_1)$ or reject
reject if $(n'_C \neq n_C)$ or $(S' \neq S)$
Client/Server Authentication (CS) exchange
$t_C \leftarrow \mathrm{now}()$
[optional: $k^*_{CS} \leftarrow\$ \mathcal{K}^*$]
$c_3 \leftarrow\$ \mathrm{Enc}_{k_{CS}}(C, t_C[, k^*_{CS}])$
c_2, c_3
$(k_{CS}, t_{KAS}, C) \leftarrow \mathrm{Dec}_{k_S}(c_2)$ or reject
$(C', t_C[, k^*_{CS}]) \leftarrow \mathrm{Dec}_{k_{CS}}(c_3)$ or reject
reject if $\neg\mathrm{timeok}(t_{KAS}, t_C; \mathrm{now}())$
reject if $C \neq C'$
$c_4 \leftarrow\$ \mathrm{Enc}_{k_{CS}}(t_C)$
accept [and output k^*_{CS}]
c_4
$t'_C \leftarrow \mathrm{Dec}_{k_{CS}}(c_4)$ or reject
reject if $t'_C \neq t_C$
accept [and output k^*_{CS}]

Figure 2: The 3-party form of Kerberos v5, protocol $\mathtt{3K} = \mathtt{3K}[\Pi, \lambda]$ constructed from authenticated encryption scheme Π and nonce length λ; $[k^*_{CS}]$ denotes optional sub-session key mode.

Limitations. Our work focuses on the two-exchange/three-party mode of Kerberos with symmetric key authentication; it excludes the optional three-exchange/four-party mode, as well as the public key modes. Our proof for authentication of the three-party mode is already lengthy, requiring 19 game hops across 2 cases. Some other works have looked at the four-party mode, but these also have limitations as noted above. While Backes et al. and Blanchet et al. [BJST08] both analyze the three-exchange/four-party mode, they only analyze each exchange separately but do not say how to compose results on the three exchanges to obtain security against an adversary who can arbitrarily attack any stage of the protocol. Directly in the computational setting, it would be desirable to use a composable approach like Brzuska and Jacobsen [BJ17] applied to the EAP protocol (which is composed of three separate sub-protocols), the individual exchanges of Kerberos do not seem quite substantial enough on their own to extract as sub-protocols which can then be composed. We leave a suitable abstraction in the computational setting as future work.

2 The Kerberos protocol

The 3-party Kerberos protocol (which is the focus of the remainder of this paper) is built from an authenticated encryption scheme $\Pi = (\mathrm{Enc}, \mathrm{Dec})$. In this mode of Kerberos, a central *Kerberos Authentication Server (KAS)* shares long-term symmetric keys with all clients and servers in the network. Any pair of client C and server S can mutually authenticate and agree on a symmetric key k_{CS} with the assistance of the KAS using the protocol flow depicted in Figure 2. In the standard [KN93, NYHR05], the Authentication Service (AS) exchange consists of steps 1–3 below, and the

Client/Server Authentication (CS) exchange consists of steps 4–6 below.

(1) To initiate, client C chooses a fresh nonce n_C and sends its identity C, the identity of its desired peer server S, and the nonce to the KAS. This message is not protected.

(2) On receiving (C, S, n_C), the KAS chooses a fresh session key k_{CS} and constructs two ciphertexts, one for the client (c_1) and one for the server (c_2). Both contain the session key k_{CS}, and the time t_{KAS} when the KAS generated the message, and the identity of each party's intended peer. The client's nonce is echoed back to the client to check freshness. Each is encrypted and authenticated using the long-term key each party shares with the KAS.

(3) The KAS forwards the two ciphertexts to C, who decrypts and verifies c_1, and checks the nonce and server identity match the (unprotected) request made by the client.

(4) The client C proceeds to construct a ciphertext c_3 for the server, which contains the client's identity and current time. This message is encrypted under the session key k_{CS}, and is meant to demonstrate C's possession of the key k_{CS} to S. The client forwards c_2 along with c_3 to the server.

(5) The server S decrypts and verifies both ciphertexts, first c_2 to retrieve k_{CS}, and then c_3 using this key. If either of these fails, the server rejects. Otherwise, it checks if the peer identities contained in c_2 and c_3 match, and if the two timestamps contained in c_2 and c_3 are "fresh enough" by calling some function timeok(). If so, the server accepts the authentication of C as valid, and generates a ciphertext c_4 for C the timestamp t_C received in message c_3 using the session key k_{CS} to also prove possession of k_{CS}.

(6) The client decrypts and verifies c_4 and checks if the returned time value matches the one it sent. If so, the client accepts the authentication of S as valid.

The timestamps play a central role in enabling the server to reject most replay attacks without maintaining a large list of previous sessions. Our model does not prevent replay attacks against the server, so freshness of timestamps does not play a role in our analysis. We discuss some aspects of modelling time in computational models in Appendix B.

In an optional mode, the parties can establish a *sub-session key*, separate from the session key. It can be established either (a) by the client picking a fresh random key k^*_{CS} and encrypting it as part of the ciphertext c_3 sent to the server; or (b) the server picking k^*_{CS} and encrypting it as part of c_4 sent to the client. Figure 2 shows the first case for simplicity.

Threat model. Kerberos is designed to be operated in a network environment where the adversary controls the communication channel (so it can drop, delay, reorder, change, or create messages) and can initiate client instances. Kerberos is designed to be secure against compromise of individual clients and servers, but not against the compromise of the trusted KAS. The primary security goal, which we formalize in the next section, is mutual authentication of the client and server. A secondary goal, which we address in Section 6, is establishment of a secret session key. In Appendix A, we discuss some attack scenarios which Kerberos is not designed to protect against.

3 Security model for 3-party authentication

Kerberos has two use cases: it can be used as a symmetric authentication protocol (in which case the session key k_{CS} is only used within the protocol), or as a means to distribute an authenticated key to client and server for further, non-specified use. To analyze the security of each of these settings, our security models start from a common execution model, but diverge when it comes to the specific goal of the adversary. Here we focus on 3-party authentication; we consider security of the session key in Section 6.

3.1 Execution model

We consider the execution of a 3-party protocol in an environment with $n = n_{\mathsf{clnts}} + n_{\mathsf{srvrs}} + 1$ parties, of which n_{clnts} are clients, n_{srvrs} are servers, and there is a single authentication server. Our model assumes a stateless authentication server, meaning that interactions with it consist of a single query and response, without any ongoing session state. Our model focuses on symmetric authentication.

Protocol. A 3-party protocol consists of four algorithms:

- $\mathrm{KG}() \xrightarrow{\$} k$: A probabilistic long-term key generation algorithm that generates a random symmetric key k.
- $\mathrm{Run}_{\mathrm{AS}}(C, S, m) \xrightarrow{\$} (m', \mathsf{cid})$: A probabilistic algorithm executed by the authentication server, which takes as input identifiers of a client C and server S, as well as a message m (and implicitly long-term keys of all parties), and outputs a message m' and a contributive identifier cid.
- $\mathrm{Run}_{\mathrm{C}}(m, \pi) \xrightarrow{\$} (m', \pi')$ A probabilistic algorithm executed by the client, which takes as input message m and instance state π (and implicitly the party's long-term key), and outputs an outgoing message m' (possibly empty $\perp$), and an updated instance state π'. As the client is the initiator, the protocol is started by running $\mathrm{Run}_{\mathrm{C}}$ with m as the client's configuration data.
- $\mathrm{Run}_{\mathrm{S}}(m, \pi) \xrightarrow{\$} (m', \pi')$ A probabilistic algorithm executed by the server, analogous to $\mathrm{Run}_{\mathrm{C}}$.

Parties and instances. Each client or server P shares with the authentication server a long-term key k_P which is established at the beginning of the experiment. Each party may run up to n_{sess} protocol instances (executed concurrently or sequentially) with their own local state. The ith instance of a party P is represented by π^i_P, which is a collection of variables containing the local state. P is called the *owner* of instance π^i_P. The following state variables represent "real" protocol values which would typically be found in implementations of a protocol:

- $\pi^i_P.\mathsf{status}$: The status of the instance π^i_P. For client and server instances, this could be the uninitialized value $\perp$, or $\mathsf{accepted}$ or $\mathsf{rejected}$. For client instances, this could also be a counter which records which stage of execution the client instance is at.
- $\pi^i_P.\mathsf{pid}$: The identifier of the intended peer of this instance.
- $\pi^i_P.k$: The session key k established in this instance.
- Additional variables maintained during the execution of the protocol. (In Kerberos, these will include a nonce $\pi^i_P.n$ and a time $\pi^i_P.t$.)

Additionally, some state variables are maintained in the security experiment to assist with modelling:

- $\pi^i_P.\mathsf{cid}$: A *contributive* identifier, used in the freshness definition as explained below.
- $\pi^i_P.\mathsf{sid}$: A *session* identifier, used in the authentication definition as explained below.
- $\pi^i_P.\mathsf{revealed}$: A flag indicating if the adversary has revealed the session key via the Reveal oracle.

The protocol must specify what values comprise the contributive and session identifiers, and in particular will be set by $\mathrm{Run}_{\mathrm{C}}$ / $\mathrm{Run}_{\mathrm{S}}$.

Note that when working with contributive and session identifiers we use the notation $\|$ to denote unambiguous concatenation and $\mathsf{sid}[j]$ to denote the jth component.

Adversarial interaction. The adversary can interact with the system using the oracles specified in Figure 3. The following oracles model execution of honest parties:

- $\mathsf{Send}_{\mathsf{AS}}(C, S, m)$: Models the (stateless) execution of the authentication server to establish a session between parties C and S using message m. During its execution, it maintains a list cids of contributive identifiers which is used in the freshness definition.
- $\mathsf{Send}(P, i, m)$: Models execution of party P's ith instance, π_P^i, (where P is a client or server) with input message m; the output message (if any) is returned to the adversary, and the instance variables π_P^i may be updated.

Using these queries, the adversary can control the delivery of all messages, including dropping, delaying, reordering, changing, or creating messages.

The adversary can also learn some secret information from parties, either long-term keys or session keys, which is modelled by the following oracles:

- $\mathsf{Corrupt}(P)$: The adversary learns the long-term key k_P of party P, which is then marked as corrupted ($\mathsf{corrupted}_P \leftarrow \mathsf{true}$).
- $\mathsf{Reveal}(P, i)$: The adversary learns the session key of party P's ith instance, namely $\pi_P^i.k$, which is then marked as revealed ($\pi_P^i.\mathsf{revealed} \leftarrow \mathsf{true}$).

3.2 Security experiment for 3-party authentication

Figures 3 shows the security experiment 3-auth for mutual authentication of a client and server, via an authentication server.

The idea of the experiment is as follows. The main experiment ($\mathrm{Exp}^{\mathsf{3\text{-}auth}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{M},\mathsf{F}}(\mathcal{A})$) sets up the execution environment (generating long-term keys that each client and server shares with the authentication server), then runs the adversary $\mathcal{A}$. The adversary can interact with the authentication server, clients, and servers using the corresponding $\mathsf{Send}_{\mathsf{AS}}$ and Send oracles

The adversary is deemed to have won the experiment if it makes a client or server instance *maliciously accept*, as defined by the malicious-accept predicate. (We sometimes called the maliciously accepting instance the "target instance".) This means an instance has accepted without a peer instance, violating the general idea that "authentication" means existence of a peer who agrees on the messages sent and received; formally, this agreement is modelled by the match predicate M, and may be specific to the protocol being analyzed. One example is the matching conversations notion from Bellare and Rogaway [BR94].

Since the adversary can compromise certain secrets, we must restrict the winning condition to exclude scenarios where the adversary has revealed secrets that let it trivially impersonate the parties and trivially win the game. The malicious-accept predicate restricts the adversary from learning the long-term secrets of either party involved in the target instance. Via the freshness predicate F, it may also restrict the adversary from learning other secrets, and may be specific to the protocol being analyzed. See Section 4 for details of the predicates we use for analyzing Kerberos.

Having defined the experiment, we can define the corresponding advantage for an adversary algorithm $\mathcal{A}$:

$$\mathrm{Adv}^{\mathsf{3\text{-}auth}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{M},\mathsf{F}}(\mathcal{A}) = \Pr\left[\mathrm{Exp}^{\mathsf{3\text{-}auth}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{M},\mathsf{F}}(\mathcal{A}) \Rightarrow \mathsf{true}\right] .$$

```
$\mathrm{Exp}^{\textsf{3-auth}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{M},\mathsf{F}}(\mathcal{A})$
 1: // generate long-term keys
 2: for $P \in \mathcal{C} \cup \mathcal{S}$ do
 3:     $k_P \leftarrow\$ \Pi.\mathrm{KG}()$
 4: // run the experiment
 5: $\mathcal{A}^{\mathsf{Send}_{\mathsf{AS}},\mathsf{Send},\mathsf{Reveal},\mathsf{Corrupt}}()$
 6: // winning condition
 7: return true iff $\exists\ P \in \mathcal{C} \cup \mathcal{S}, i \in [n_{\mathsf{sess}}]$
    such that $\mathsf{malicious\text{-}accept}_{\mathsf{M},\mathsf{F}}(P, i)$)

$\mathsf{malicious\text{-}accept}_{\mathsf{M},\mathsf{F}}(A, i)$:
 1: $B \leftarrow \pi^i_A.\mathsf{pid}$
 2: return true iff
 3:         // adversary wins if the session accepted
 4:         $(\pi^i_A.\mathsf{status} = \mathsf{accepted})$
 5:         // ... and the session owner was uncorrupted
 6:     $\wedge\ \neg\mathsf{corrupted}_A$
 7:         // ... and the intended peer was uncorrupted
 8:     $\wedge\ \neg\mathsf{corrupted}_B$
 9:         // ... and the session is "fresh"
10:     $\wedge\ \mathsf{F}(A, i)$
11:         // ... and there is no peer session with a matching transcript.
12:     $\wedge\ \neg(\exists j \in [n_{\mathsf{sess}}] : \mathsf{M}(A, i, B, j))$

$\mathsf{Send}_{\mathsf{AS}}(C, S, m)$:
 1: // run the authentication server's code
 2: $(m', \mathsf{cid}) \leftarrow\$ \Pi.\mathrm{Run}_{\mathrm{AS}}(C, S, m)$
 3: // keep list of contributive identifiers for freshness predicate
 4: $\mathsf{cids} \leftarrow \mathsf{cids} \| \mathsf{cid}$
 5: return $m'$

$\mathsf{Send}(P, i, m)$:
 1: // run the party's code
 2: if $P \in \mathcal{C}$ then
 3:     $(m', \pi^i_P) \leftarrow\$ \mathrm{Run}_{\mathrm{C}}(m, \pi^i_P)$
 4: else
 5:     $(m', \pi^i_P) \leftarrow\$ \mathrm{Run}_{\mathrm{S}}(m, \pi^i_P)$
 6: return $m'$

$\mathsf{Reveal}(P, i)$:
 1: // only reveal keys in sessions that have not rejected
 2: if $\pi^i_P.\mathsf{status} \neq \mathsf{rejected}$ then
 3:     $\pi^i_P.\mathsf{revealed} \leftarrow \mathsf{true}$
 4:     return $\pi^i_P.k$

$\mathsf{Corrupt}(P)$:
 1: $\mathsf{corrupted}_P \leftarrow \mathsf{true}$
 2: return $k_P$
```

Figure 3: Authentication security experiment for 3-party protocols

4 Instantiating Kerberos in the model

The M and F predicates we use to analyze 3-party Kerberos are defined in Figure 4. Our M predicate is based on matching conversations, but only for a subset of the transcript. Our F predicate restricts the adversary from learning the session key of the target instance or of related instances.

Figure 5 shows 3-party Kerberos from Figure 2 instantiated as a protocol in our formalism. Specifically, $\mathtt{3K} = \mathtt{3K}[\Pi, \lambda]$ is constructed from an authenticated encryption scheme Π and a nonce length λ. (We rely on standard definitions for authenticated encryption schemes [BN08], which we include in Appendix C.)

4.1 Discussion of security model

There are certain subtleties in our security experiment for 3-party authentication that merit further discussion, especially when compared with typical authenticated key exchange security definitions.

Following recent works like Fischlin and Günther [FG14], we distinguish between the *contributive* identifier and the *session* identifier. The session identifier is used in the M predicate to indicate which other instances must exist when an instance accepts. We cannot rely simply on matching conversations on the full transcript: in Kerberos, the adversary can send a different c_2 to the client and server yet have them both accept with the same session key, so mismatching on c_2 should not violate authentication. Thus in Kerberos authentication should only demand that clients and servers match on c_3 and c_4: this is captured by the *session identifier* sid. (As is typical, the adversary could drop the final c_4 from the client to the server, so we allow that possibility in the M predicate.)

The contributive identifier is used in the F predicate to indicate which other instances' session

```
$\underline{\mathsf{M}(A,i,B,j)}$: // session matching for Kerberos

1: if $A \in \mathcal{C}, B \in \mathcal{S}$ then
2:     return $(\pi_B^j.\mathsf{sid} = \pi_A^i.\mathsf{sid})$
3: else if $A \in \mathcal{S}, B \in \mathcal{C}$ then
4:     return $(\pi_A^i.\mathsf{sid}[0] = \pi_B^j.\mathsf{sid}[0])$
```

```
$\underline{\mathsf{F}(A,i)}$: // session freshness for Kerberos

1: if $A \in \mathcal{C}$ then
2:     // a client session is unfresh if the session key was revealed at a session at the originally intended peer server that has a matching contributive identifier
3:     if $\exists\ B, j\ :\ (A, B, \pi_A^i.cid, \pi_B^j.cid) \in \mathsf{cids} \wedge \pi_B^j.\mathsf{revealed}$ then
4:         return false
5:     // ... or the session key was revealed at the session itself
6:     return $\neg\ \pi_A^i.\mathsf{revealed}$
7: else if $A \in \mathcal{S}$ then
8:     // a server session is unfresh if the session key was revealed at a session at the originally intended peer client that has a matching contributive identifier
9:     if $\exists\ B, j\ :\ (B, A, \pi_B^j.cid, \pi_A^i.cid) \in \mathsf{cids} \wedge \pi_B^j.\mathsf{revealed}$ then
10:        return false
11:    // ... or the session key was revealed at any of this party's sessions where it used this contributive identifier
12:    return $\neg(\exists\ j : \pi_A^i.\mathsf{cid} = \pi_A^j.\mathsf{cid} \wedge \pi_A^j.\mathsf{revealed})$
```

Figure 4: Matching and freshness predicates for analysis of 3-party Kerberos

$\text{Run}_{\text{AS}}(C, S, n)$:

```
1: // generate a random key and encrypt it for the client and server
2: $k \leftarrow_\$ \mathcal{K}$
3: $t \leftarrow \text{now}()$
4: $c_1 \leftarrow_\$ \Pi.\text{Enc}_{k_C}(k, n, t, S)$
5: $c_2 \leftarrow_\$ \Pi.\text{Enc}_{k_S}(k, t, C)$
6: return $(m' = (c_1, c_2), \mathsf{cid} = (C, S, c_1, c_2))$
```

$\text{Run}_{\text{S}}(m, \pi_S^i)$:

```
1: // instance must be unused
2: if $\pi_S^i.\mathsf{status} \neq \perp$ then
3:     return $(m' = \perp, \pi_S^i)$
4: parse $m$ as $(c_2, c_3)$
5: $(\pi_S^i.k, t_{KAS}, C) \leftarrow \Pi.\text{Dec}_{k_S}(c_2)$ // decrypt $c_2$ if possible
6: if Dec returned $\perp$ then
7:     $\pi_S^i.\mathsf{status} \leftarrow \mathsf{rejected}$
8:     return $(m' = \perp, \pi_S^i)$
9: $(C, t_C) \leftarrow \Pi.\text{Dec}_{\pi_S^i.k}(c_3)$ // decrypt $c_3$ if possible
10: if Dec returned $\perp$ then
11:     $\pi_S^i.\mathsf{status} \leftarrow \mathsf{rejected}$
12:     return $(m' = \perp, \pi_S^i)$
13: $t \leftarrow \text{now}()$
14: if $\text{timeok}(t_{KAS}, t_C; t)$ then
15:     $c_4 \leftarrow_\$ \Pi.\text{Enc}_{\pi_S^i.k}(t_C)$ // generate outgoing ciphertext
16:     // accept and record identifiers
17:     $\pi_S^i.\mathsf{status} \leftarrow \mathsf{accepted}$
18:     $\pi_S^i.\mathsf{cid} \leftarrow c_2$
19:     $\pi_S^i.\mathsf{sid} \leftarrow c_3 \| c_4$
20:     $\pi_S^i.\mathsf{pid} \leftarrow C$
21:     return $(m' = c_4, \pi_S^i)$
22: else
23:     $\pi_S^i.\mathsf{status} \leftarrow \mathsf{rejected}$
24:     return $(m' = \perp, \pi_S^i)$
```

$\text{Run}_{\text{C}}(m, \pi_C^i)$ where $\pi_C^i.\mathsf{status} = \perp$: // *start client instance*

```
1: parse $m$ as $S$
2: $\pi_C^i.n \leftarrow_\$ \{0,1\}^\lambda$ // generate nonce
3: $\pi_C^i.\mathsf{status} \leftarrow 1$ // record instance state
4: $\pi_C^i.\mathsf{pid} \leftarrow S$
5: return $(m' = (C, S, \pi_C^i.n), \pi_C^i)$
```

$\text{Run}_{\text{C}}(m, \pi_C^i)$ where $\pi_C^i.\mathsf{status} = 1$: // *process KAS message*

```
1: parse $m$ as $(c_1, c_2)$
2: $(\pi_C^i.k, n', t_{KAS}, S') \leftarrow \Pi.\text{Dec}_{k_C}(c_1)$ // decrypt $c_1$ if possible
3: if Dec returned $\perp$ then
4:     $\pi_C^i.\mathsf{status} \leftarrow \mathsf{rejected}$
5:     return $(m' = \perp, \pi_C^i)$
6: // check received data based on Kerberos protocol specification
7: if $(\pi_C^i.n = n') \wedge (\pi_C^i.\mathsf{pid} = S')$ then
8:     // generate outgoing ciphertext
9:     $\pi_C^i.t \leftarrow \text{now}()$
10:    $c_3 \leftarrow_\$ \Pi.\text{Enc}_{\pi_C^i.k}(C, \pi_C^i.t)$
11:    $\pi_C^i.\mathsf{status} \leftarrow 2$ // update instance state
12:    $\pi_C^i.\mathsf{cid} \leftarrow c_1$ // record identifiers
13:    $\pi_C^i.\mathsf{sid} \leftarrow c_3$
14:    return $(m' = (c_2, c_3), \pi_C^i)$
15: else
16:    $\pi_C^i.\mathsf{status} \leftarrow \mathsf{rejected}$
17:    return $(m' = \perp, \pi_C^i)$
```

$\text{Run}_{\text{C}}(m, \pi_C^i)$ where $\pi_C^i.\mathsf{status} = 2$: // *process server message*

```
1: parse $m$ as $c_4$
2: $t' \leftarrow \Pi.\text{Dec}_{\pi_C^i.k}(c_4)$ // decrypt $c_4$ if possible
3: if Dec returned $\perp$ then
4:     $\pi_C^i.\mathsf{status} \leftarrow \mathsf{rejected}$
5:     return $(m' = \perp, \pi_C^i)$
6: // check received data based on Kerberos protocol specification
7: if $t' = \pi_C^i.t$ then
8:     // accept and update identifiers
9:     $\pi_C^i.\mathsf{status} \leftarrow \mathsf{accepted}$
10:    $\pi_C^i.\mathsf{sid} \leftarrow \pi_C^i.\mathsf{sid} \| c_4$
11: else
12:    $\pi_C^i.\mathsf{status} \leftarrow \mathsf{rejected}$
13: return $(m' = \perp, \pi_C^i)$
```

Figure 5: Specification of 3-party Kerberos protocol $\mathtt{3K}[\Pi, \lambda]$ with symmetric encryption scheme Π and nonce length λ.

keys are related to a particular instance. In Kerberos, session keys are established by the Kerberos authentication server, and distributed in the ciphertexts c_1 and c_2. Therefore, any instances involving c_1 or c_2 use the same session key, and thus cannot be revealed without compromising security of the target instance. For modelling purposes, we depend on the authentication maintaining a list cids of contributive identifiers that records which c_1 and c_2 are related; we have no other way of tracking this because there is no requirement that either the client or the server see an authentic version of the KAS message intended for the other party. For client instances, there should be at most one instance using the same c_1, so we only have to worry about one instance's session key being revealed. For server instances, however, there is a priori no replay protection, and thus there could be multiple instances using the same c_2, so we have to worry about any of these session keys being revealed, hence the expression on line 12 of the F predicate.

5 Security proof for 3-party authentication of Kerberos

Theorem 1. *If Π is an $\mathsf{IND\text{-}CPA}$- and $\mathsf{INT\text{-}CTXT}$-secure authenticated encryption scheme, and the nonce size λ is sufficiently large, then $\mathsf{3K}[\Pi, \lambda]$ is a secure 3-party authentication protocol, in the sense of Section 3.2, with M and F predicates from Figure 4. More specifically, for any adversary algorithm $\mathcal{A}$, there exist algorithms $B_{C3}, \ldots$ (with approximately the same runtime as $\mathcal{A}$), described in the proof, such that*

$$\begin{aligned}
\mathrm{Adv}^{\mathsf{3\text{-}auth}}_{\mathsf{3K}[\Pi,\lambda],\mathcal{C},\mathcal{S},n_{\mathsf{sess}},M,F}(\mathcal{A}) \leq{}& \frac{(n_{\mathsf{clnts}} \cdot n_{\mathsf{sess}})^2}{2^\lambda} \\
&+ n_{\mathsf{srvrs}}\Big(\mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{C3}) + \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{B}_{C4}) \\
&\qquad + n_{\mathsf{clnts}}\Big(\mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{C6}) + \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{B}_{C7}) \\
&\qquad\qquad + n_{\mathsf{kas}} \cdot \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{C9})\Big)\Big) \\
&+ n_{\mathsf{clnts}}\Big(\mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S3}) + \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{B}_{S4}) \\
&\qquad + n_{\mathsf{srvrs}}\Big(\mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S6}) + \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{B}_{S7}) \\
&\qquad\qquad + n_{\mathsf{kas}}\big(\mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S9}) + \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S10})\big)\Big)\Big),
\end{aligned}$$

where n_{kas} is the number of queries made by the adversary to the KAS.

We use the standard definitions of $\mathsf{IND\text{-}CPA}$ and $\mathsf{INT\text{-}CTXT}$ security [BN08].

Proof. The proof is divided into two cases—whether (case C) client-to-server authentication or (case S) server-to-client authentication is broken—and each case proceeds by a sequence of games. Table 1 summarizes the proof structure. We let S_i denote the event that the experiment outputs true in game G_i.

Game G_0

This is the original experiment, $\mathrm{Exp}^{\mathsf{3\text{-}auth}}_{\mathsf{3K}[\Pi,\lambda],\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{M},\mathsf{F}}(\mathcal{A})$. By definition,

$$\mathrm{Adv}^{\mathsf{3\text{-}auth}}_{\mathsf{3K}[\Pi,\lambda],\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{M},\mathsf{F}}(\mathcal{A}) = \Pr[S_0] \ .$$

Table 1: Proof structure for Theorem 1, proof of 3-party-authentication security of 3-party Kerberos $\mathtt{3K}[\Pi, \lambda]$

Game	Change from previous game	Argument based on
G_0	original 3-auth experiment for $\mathtt{3K}[\Pi, \lambda]$	
Case S: server-to-client authentication (assume first malicious accept is at a client)		
G_{S1}	exclude nonce collisions	birthday bound
G_{S2}	guess first client C^* to maliciously accept	guessing probability
G_{S3}	abort if C^* accepts any forged c_1 ciphertext	INT-CTXT reduction
G_{S4}	replace session keys k in c_1 ciphertexts to C^* with random keys	IND-CPA reduction
G_{S5}	guess peer S^* of first client to maliciously accept	guessing probability
G_{S6}	abort if S^* accepts any forged c_2 ciphertext	INT-CTXT reduction
G_{S7}	replace session keys k in c_2 ciphertexts to S^* with random keys	IND-CPA reduction
G_{S8}	guess which KAS call generated the session key for the malicious session	guessing probability
G_{S9}	abort if S^* accepts any forged c_3 ciphertext	INT-CTXT reduction
G_{S10}	abort if C^* accepts any forged c_4 ciphertext	INT-CTXT reduction
Case C: client-to-server authentication (assume first malicious accept is at a server)		
G_{C1}	exclude nonce collisions	birthday bound
G_{C2}	guess first server S^* to maliciously accept	guessing probability
G_{C3}	abort if S^* accepts any forged c_2 ciphertext	INT-CTXT reduction
G_{C4}	replace session keys k in c_2 ciphertexts to S^* with random keys	IND-CPA reduction
G_{C5}	guess peer C^* of first server to maliciously accept	guessing probability
G_{C6}	abort if C^* accepts any forged c_1 ciphertext	INT-CTXT reduction
G_{C7}	replace session keys k in c_1 ciphertexts to C^* with random keys	IND-CPA reduction
G_{C8}	guess which KAS call generated the session key for the malicious session	guessing probability
G_{C9}	abort if S^* accepts any forged c_3 ciphertext	INT-CTXT reduction

Case C: Client-to-server authentication

In this case, we assume that client authentication is broken first, namely, the first party to maliciously accept is a server.

Game G_{C1}

Difference from previous game: We exclude nonce collisions: the game aborts if the same nonce value is chosen twice by any honest client.

Analysis of abort event: The number of nonces chosen throughout the game is at most $n_{\mathsf{clnts}} \cdot n_{\mathsf{sess}}$, and each nonce is chosen from a set of size 2^λ. Thus,

$$|\Pr[S_0] - \Pr[S_{C1}]| \leq \frac{(n_{\mathsf{clnts}} \cdot n_{\mathsf{sess}})^2}{2^\lambda} \ .$$

Game G_{C2}

Difference from previous game: The game guesses a value $S^* \in \mathcal{S}$, and aborts if S^* is not the first server to maliciously accept.

Analysis of abort event: The guess is correct with probability $1/|\mathcal{S}|$, thus

$$\Pr[S_{C1}] \leq n_{\mathsf{srvrs}} \Pr[S_{C2}] \ .$$

From this point on, we can assume there exists some i such that $\mathsf{malicious\text{-}accept}(S^*, i)$ is true.

Game G_{C3}

Difference from previous game: This game aborts if there exists any instance $\pi^i_{S^*}$ which does not reject when it receives a c_2 that was not output by a call to $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_2 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{C3}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{C2}, except as follows.

Reduction description: $\mathcal{B}_{C3}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{S^*} for S^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$ queries, it generates c_2 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and then stores c_2 and the corresponding plaintext in a list. When it needs to answer $\mathsf{Send}(S^*, i, (c_2, \cdot))$ queries, it passes all c_2 to the Dec^* oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger; if c_2 was in the list stored during the $\mathsf{Send}_{\mathsf{AS}}$ calls, it uses the corresponding plaintext and continues as specified, otherwise it rejects.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_2 in $\mathsf{Send}(S^*, \dots)$ is correct. Reveal queries: no changes in how these are answered. $\mathsf{Corrupt}$ queries: we cannot answer $\mathsf{Corrupt}(S^*)$ queries since the simulation does not know k_{S^*}, but we don't need to: since there is some i such that $\mathsf{malicious\text{-}accept}(S^*, i)$, we must have that S^* has not been corrupted.

Winning: When the abort event occurs, some $\mathsf{Send}(S^*, i, (c_2, \cdot))$ has received a c_2 that was not output by $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$. Thus, none of $\mathcal{B}_{C3}$'s calls to its $\mathsf{INT\text{-}CTXT}$ Enc oracle ever returned c_2. But since this instance did not reject, it must be that $\mathcal{B}_{C3}$ call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c_2 did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{C3}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{C2}] - \Pr[S_{C3}]| \le \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{C3}) \ .$$

Game G_{C4}

Differences from previous game: In $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$ queries, generate a random k', and use k' in the generation of c_2, rather than k; store (c_2, k) in a list. In $\mathsf{Send}(S^*, i, (c_2, \cdot))$ queries where (c_2, k) is in the aforementioned list for some k, proceed using k as the key. In $\mathsf{Send}(S^*, i, (c_2, \cdot))$ queries where no $(c_2, \cdot)$ entry is in the list, reject.

Analysis of game difference: We claim that any adversary that can distinguish the previous game from this one can be used to break the indistinguishability of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{C4}$ which interacts with an $\mathsf{IND\text{-}CPA}$ challenger for Π and behaves exactly as in game G_{C3}, except as follows.

Reduction description: $\mathcal{B}_{C4}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{S^*} for S^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$ queries, it picks a random k', then generates c_2 by calling the $\mathrm{Enc}(\mathrm{LR}(\cdot, \cdot))$ oracle provided by the $\mathsf{IND\text{-}CPA}$ challenger, using two plaintexts $m_0 = (k, n, C)$ and $m_1 = (k', n, C)$. It stores (c_2, k) in a list. When it needs to answer $\mathsf{Send}(S^*, i, (c_2, \cdot))$ queries, where (c_2, k) is in the list for some k, it proceeds using k as the key. When it needs to answer $\mathsf{Send}(S^*, i, (c_2, \cdot))$ queries, where no $(c_2, \cdot)$ is in the list, it rejects.

Simulation correctness: When the bit b in the $\mathsf{IND\text{-}CPA}$ challenger is 0, $\mathcal{B}_{C4}$ exactly simulates game G_{C3}. When the bit b is 1, $\mathcal{B}_{C4}$ exactly simulates game G_{C4}. Thus,

$$|\Pr[S_{C3}] - \Pr[S_{C4}]| \le \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{B}_{C4}) \ .$$

Game G_{C5}

Difference from previous game: The game guesses a value $C^* \in \mathcal{C}$, and aborts if C^* is not the alleged peer in the first maliciously accepting server session.

Analysis of abort event: The guess is correct with probability $1/|\mathcal{C}|$, thus

$$\Pr[S_{C4}] \leq n_{\mathsf{clnts}} \Pr[S_{C5}] \ .$$

Game G_{C6}

Difference from previous game: This game aborts if there exists any instance $\pi^j_{C^*}$ which does not reject when it receives a c_1 that was not output by a call to $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_1 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{C6}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{C5}, except as follows.

Reduction description: $\mathcal{B}_{C6}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{C^*} for C^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$ queries, it generates c_1 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and then stores c_1 and the corresponding plaintext in a list. When it needs to answer $\mathsf{Send}(C^*, j, (c_1, \cdot))$ queries, it passes all c_1 to the Dec^* oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger; if c_1 was in the list stored during the $\mathsf{Send}_{\mathsf{AS}}$ calls, it uses the corresponding plaintext and continues as specified, otherwise it rejects.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_1 in $\mathsf{Send}(C^*, \dots)$ is correct. Reveal queries: no changes in how these are answered. $\mathsf{Corrupt}$ queries: we cannot answer $\mathsf{Corrupt}(C^*)$ queries since the simulation does not know k_{C^*}, but we don't need to: since there is some i such that $\mathsf{malicious\text{-}accept}(S^*, i)$, and $\pi^i_{S^*}.\mathsf{pid} = C^*$, we must have that C^* has not been corrupted.

Winning: When the abort event occurs, some $\mathsf{Send}(C^*, j, (c_1, \cdot))$ has received a c_1 that was not output by $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$. Thus, none of $\mathcal{B}_{C6}$'s calls to its $\mathsf{INT\text{-}CTXT}$ Enc oracle ever returned c_1. But since this instance did not reject, it must be that $\mathcal{B}_{C6}$ call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c_1 did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{C6}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{C5}] - \Pr[S_{C6}]| \leq \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{C6}) \ .$$

Game G_{C7}

Differences from previous game: In $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$ queries, generate a random k'', and use k'' in the generation of c_1, rather than k; store (c_1, k) in a list. (The list in this game is separate from the list in game G_{C4}.) In $\mathsf{Send}(C^*, j, (c_1, \cdot))$ queries where (c_1, k) is in the aforementioned list for some k, proceed using k as the key. In $\mathsf{Send}(C^*, i, (c_1, \cdot))$ queries where no $(c_1, \cdot)$ entry is in the list, reject.

Analysis of game difference: We claim that any adversary that can distinguish the previous game from this one can be used to break the indistinguishability of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{C7}$ which interacts with an $\mathsf{IND\text{-}CPA}$ challenger for Π and behaves exactly as in game G_{C6}, except as follows.

Reduction description: $\mathcal{B}_{C7}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{C^*} for C^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$ queries, it picks a random k'', then generates c_1 by calling the $\mathrm{Enc}(\mathrm{LR}(\cdot, \cdot))$ oracle provided by the $\mathsf{IND\text{-}CPA}$ challenger, using two plaintexts $m_0 = (k, n, t, S)$ and $m_1 = (k'', n, t, S)$. It stores (c_1, k) in a list.

When it needs to answer $\mathsf{Send}(C^*, j, (c_1, \cdot))$ queries, where (c_1, k) is in the list for some k, it proceeds using k as the key. When it needs to answer $\mathsf{Send}(C^*, j, (c_1, \cdot))$ queries, where no $(c_1, \cdot)$ is in the list, it rejects.

Simulation correctness: When the bit b in the $\mathsf{IND\text{-}CPA}$ challenger is 0, $\mathcal{B}_{C7}$ exactly simulates game G_{C6}. When the bit b is 1, $\mathcal{B}_{C7}$ exactly simulates game G_{C7}. Thus,

$$|\Pr[S_{C6}] - \Pr[S_{C7}]| \leq \mathrm{Adv}_{\Pi}^{\mathsf{ind\text{-}cpa}}(\mathcal{B}_{C7}) \ .$$

Game G_{C8}

Difference from previous game: The game guesses a value $\ell \in [1, n_{\mathsf{kas}}]$, and aborts if the c_2 in the first S^* session to maliciously accept was not generated by the ℓth $\mathsf{Send}_{\mathsf{AS}}(C^*, S^*, \cdot)$ call.

Analysis of abort event: The guess is correct with probability at least $1/n_{\mathsf{kas}}$, thus

$$\Pr[S_{C7}] \leq n_{\mathsf{kas}} \Pr[S_{C8}] \ .$$

Game G_{C9}

Difference from previous game: This game aborts if there exists any instance $\pi^i_{S^*}$ with $\pi^i_{S^*}.\mathsf{pid} = C^*$ that received c_3 but no C^* session $\pi^j_{C^*}$ exists with session identifier $\pi^j_{C^*}.\mathsf{sid}$ starting with c_3.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_3 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{C9}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{C8}, except as follows.

Reduction description: Let (c_1^*, c_2^*) be the outputs of the ℓth query to $\mathsf{Send}_{\mathsf{AS}}(C^*, \S^*, \cdot)$. In $\mathsf{Send}(C^*, j, (c_1^*, \cdot))$, generate c_3 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger. (Note that this happens only once since it is preceded by a nonce check, and nonces are assumed to be unique by game G_{C1}.) Store c_3 and the corresponding plaintext in a list. In $\mathsf{Send}(S^*, i, (c_2^*, c_3))$ for any c_3, pass c_3 to the $\mathsf{INT\text{-}CTXT}$ challenger's Dec^* oracle; if c_3 was not in the aforementioned list, it rejects. Otherwise, continue with the corresponding plaintext. When it needs to generate c_4, it uses the Enc oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and stores c_4 and the corresponding plaintext in a (separate) list. In $\mathsf{Send}(C^*, j, c_4)$ for which $\pi^j_{C^*}.\mathsf{cid} = c_1^*$, use the plaintext corresponding to c_4 from the list, or reject if none.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_4 in $\mathsf{Send}(C^*, \dots)$ is correct. Reveal queries: we cannot answer $\mathsf{Reveal}(C^*, j)$ where $\pi^j_{C^*}.\mathsf{cid} = c_1^*$, but we don't need to: $\mathsf{F}(S^*, i)$ requires that there was no $\mathsf{Reveal}(C^*, j)$ for any j such that $(C^*, S^*, c_1^*, c_2^*) \in \mathsf{cids}$ (i.e., for which c_1^*, c_2^* were output by a call to $\mathsf{Send}_{\mathsf{AS}}(C^*, S*, \cdot)$.) We also cannot answer $\mathsf{Reveal}(S^*, i)$ where $\pi^i_{S^*}.\mathsf{cid} = c_2^*$, but we don't need to: $\mathsf{F}(S^*, i)$ requires there was no $\mathsf{Reveal}(C^*, j)$ for any j such that $\pi^i_{C^*}.\mathsf{cid} = \pi^j_{C^*}.\mathsf{cid}$. $\mathsf{Corrupt}$ queries: no change compared to the previous game.

Winning: When the abort event occurs, some S^* instance accepted maliciously, say the one corresponding to $\mathsf{Send}(S^*, i, (c_2^*, c_3^*))$. Note that the only place that the reduction makes calls to its $\mathsf{INT\text{-}CTXT}$ challenger's Enc oracle are either in calls to $\mathsf{Send}(C^*, j, (c_1^*, \cdot))$ (where Enc is called with plaintexts consisting of 2 components), or in $\mathsf{Send}(S^*, i, (c_2^*, \cdot))$ (where Enc is called with plaintexts consisting of 1 component). Since no C^* session exists with session identifier starting with c_3^*, the (at most one non-rejecting) $\mathsf{Send}(C^*, j, (c_1^*, \cdot))$ call did not output c_3^*. This is the only place that Enc oracle queries are made for plaintexts consisting of 2 components, so c_3^* was not the output of an Enc query to the $\mathsf{INT\text{-}CTXT}$ challenger. But since this instance did not reject, it must be that $\mathcal{B}_{C9}$

call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c_3^* did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{C9}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{C8}] - \Pr[S_{C9}]| \leq \mathrm{Adv}_{\Pi}^{\mathsf{int\text{-}ctxt}}(\mathcal{B}_{C9}) \ .$$

Analysis of Game G_{C9}

This game G_{C9} aborts if any server instance maliciously accepts, thus we have that

$$\Pr[S_{C9}] = 0$$

and our sequence of games for this case terminates.

Case S: Server-to-client authentication

In this case, we assume that server authentication is broken first, namely, the first party to maliciously accept is a client.

Game G_{S1}

Difference from previous game: We exclude nonce collisions: the game aborts if the same nonce value is chosen twice by any honest client.

Analysis of abort event: The number of nonces chosen throughout the game is at most $n_{\mathsf{clnts}} \cdot n_{\mathsf{sess}}$, and each nonce is chosen from a set of size 2^λ. Thus,

$$|\Pr[S_0] - \Pr[S_{S1}]| \leq \frac{(n_{\mathsf{clnts}} \cdot n_{\mathsf{sess}})^2}{2^\lambda} \ .$$

Game G_{S2}

Difference from previous game: The game guesses a value $C^* \in \mathcal{C}$, and aborts if C^* is not the first client to maliciously accept.

Analysis of abort event: The guess is correct with probability $1/|\mathcal{C}|$, thus

$$\Pr[S_{S1}] \leq n_{\mathsf{clnts}} \Pr[S_{S2}] \ .$$

From this point on, we can assume there exists some i such that $\mathsf{malicious\text{-}accept}(C^*, i)$ is true.

Game G_{S3}

Difference from previous game: This game aborts if there exists any instance $\pi_{C^*}^i$ which does not reject when it receives a c_1 that was not output by a call to $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_1 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{S3}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{S2}, except as follows.

Reduction description: $\mathcal{B}_{S3}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{C^*} for C^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$ queries, it generates c_1 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and then stores c_1 and the corresponding plaintext in a list. When it needs to answer $\mathsf{Send}(C^*, i, (c_1, \cdot))$ queries, it passes all c_1 to the Dec^* oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger; if c_1 was in the list stored during the $\mathsf{Send}_{\mathsf{AS}}$ calls, it uses the corresponding plaintext and continues as specified, otherwise it rejects.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_1 in $\mathsf{Send}(C^*, \dots)$ is correct. Reveal queries: no changes in how these are answered. $\mathsf{Corrupt}$ queries: we cannot answer $\mathsf{Corrupt}(C^*)$ queries since the simulation does not know k_{C^*}, but we don't need to: since there is some i such that $\mathsf{malicious\text{-}accept}(C^*, i)$, we must have that C^* has not been corrupted.

Winning: When the abort event occurs, some $\mathsf{Send}(C^*, i, (c_1, \cdot))$ has received a c_1 that was not output by $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$. Thus, none of $\mathcal{B}_{S3}$'s calls to its $\mathsf{INT\text{-}CTXT}$ Enc oracle ever returned c_1. But since this instance did not reject, it must be that $\mathcal{B}_{S3}$ call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c_1 did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{S3}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{S2}] - \Pr[S_{S3}]| \leq \mathrm{Adv}_{\Pi}^{\mathsf{int\text{-}ctxt}}(\mathcal{B}_{S3}) \ .$$

Game G_{S4}

Differences from previous game: In $\mathsf{Send}_{\mathsf{AS}}(C*, \cdot, \cdot)$ queries, generate a random k', and use k' in the generation of c_1, rather than k; store (c_1, k) in a list. In $\mathsf{Send}(C^*, i, (c_1, \cdot))$ queries where (c_1, k) is in the aforementioned list for some k, proceed using k as the key. In $\mathsf{Send}(C^*, i, (c_1, \cdot))$ queries where no $(c_1, \cdot)$ entry is in the list, reject.

Analysis of game difference: We claim that any adversary that can distinguish the previous game from this one can be used to break the indistinguishability of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{S4}$ which interacts with an $\mathsf{IND\text{-}CPA}$ challenger for Π and behaves exactly as in game G_{S3}, except as follows.

Reduction description: $\mathcal{B}_{S4}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{C^*} for C^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(C^*, \cdot, \cdot)$ queries, it picks a random k', then generates c_1 by calling the $\mathrm{Enc}(\mathrm{LR}(\cdot, \cdot))$ oracle provided by the $\mathsf{IND\text{-}CPA}$ challenger, using two plaintexts $m_0 = (k, n, t, S)$ and $m_1 = (k', n, t, S)$. It stores (c_1, k) in a list. When it needs to answer $\mathsf{Send}(C^*, i, (c_1, \cdot))$ queries, where (c_1, k) is in the list for some k, it proceeds using k as the key. When it needs to answer $\mathsf{Send}(C^*, i, (c_1, \cdot))$ queries, where no $(c_1, \cdot)$ is in the list, it rejects.

Simulation correctness: When the bit b in the $\mathsf{IND\text{-}CPA}$ challenger is 0, $\mathcal{B}_{S4}$ exactly simulates game G_{S3}. When the bit b is 1, $\mathcal{B}_{S4}$ exactly simulates game G_{S4}. Thus,

$$|\Pr[S_{S3}] - \Pr[S_{S4}]| \leq \mathrm{Adv}_{\Pi}^{\mathsf{ind\text{-}cpa}}(\mathcal{B}_{S4}) \ .$$

Game G_{S5}

Difference from previous game: The game guesses a value $S^* \in \mathcal{S}$, and aborts if S^* is not the alleged peer in the first maliciously accepting client session.

Analysis of abort event: The guess is correct with probability $1/|\mathcal{S}|$, thus

$$\Pr[S_{S4}] \leq n_{\mathsf{srvrs}} \Pr[S_{S5}] \ .$$

Game G_{S6}

Difference from previous game: This game aborts if there exists any instance $\pi_{S^*}^{j}$ which does not reject when it receives a c_2 that was not output by a call to $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_2 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{S6}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{S5}, except as follows.

Reduction description: $\mathcal{B}_{S6}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{S^*} for S^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$ queries, it generates c_2 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and then stores c_2 and the corresponding plaintext in a list. When it needs to answer $\mathsf{Send}(S^*, j, (c_2, \cdot))$ queries, it passes all c_2 to the Dec^* oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger; if c_2 was in the list stored during the $\mathsf{Send}_{\mathsf{AS}}$ calls, it uses the corresponding plaintext and continues as specified, otherwise it rejects.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_2 in $\mathsf{Send}(S^*, \dots)$ is correct. Reveal queries: no changes in how these are answered. $\mathsf{Corrupt}$ queries: we cannot answer $\mathsf{Corrupt}(S^*)$ queries since the simulation does not know k_{S^*}, but we don't need to: since there is some i such that $\mathsf{malicious\text{-}accept}(C^*, i)$, and $\pi^i_{C^*}.\mathsf{pid} = S^*$, we must have that S^* has not been corrupted.

Winning: When the abort event occurs, some $\mathsf{Send}(S^*, j, (c_2, \cdot))$ has received a c_2 that was not output by $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$. Thus, none of $\mathcal{B}_{S6}$'s calls to its $\mathsf{INT\text{-}CTXT}$ Enc oracle ever returned c_2. But since this instance did not reject, it must be that $\mathcal{B}_{S6}$ call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c_2 did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{S6}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{S5}] - \Pr[S_{S6}]| \leq \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S6}) \ .$$

Game G_{S7}

Differences from previous game: In $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$ queries, generate a random k'', and use k'' in the generation of c_2, rather than k; store (c_2, k) in a list. (The list in this game is separate from the list in game G_{S4}.) In $\mathsf{Send}(S^*, j, (c_2, \cdot))$ queries where (c_2, k) is in the aforementioned list for some k, proceed using k as the key. In $\mathsf{Send}(S^*, j, (c_2, \cdot))$ queries where no $(c_2, \cdot)$ entry is in the list, reject.

Analysis of game difference: We claim that any adversary that can distinguish the previous game from this one can be used to break the indistinguishability of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{S7}$ which interacts with an $\mathsf{IND\text{-}CPA}$ challenger for Π and behaves exactly as in game G_{S6}, except as follows.

Reduction description: $\mathcal{B}_{S7}$ generates all long-term keys as specified in the game, except it does not generate the long-term key k_{S^*} for S^*. When it needs to answer $\mathsf{Send}_{\mathsf{AS}}(\cdot, S^*, \cdot)$ queries, it picks a random k'', then generates c_2 by calling the $\mathrm{Enc}(\mathrm{LR}(\cdot, \cdot))$ oracle provided by the $\mathsf{IND\text{-}CPA}$ challenger, using two plaintexts $m_0 = (k, t, C)$ and $m_1 = (k'', t, C)$. It stores (c_2, k) in a list. When it needs to answer $\mathsf{Send}(S^*, j, (c_2, \cdot))$ queries, where (c_2, k) is in the list for some k, it proceeds using k as the key. When it needs to answer $\mathsf{Send}(S^*, j, (c_2, \cdot))$ queries, where no $(c_2, \cdot)$ is in the list, it rejects.

Simulation correctness: When the bit b in the $\mathsf{IND\text{-}CPA}$ challenger is 0, $\mathcal{B}_{S7}$ exactly simulates game G_{S6}. When the bit b is 1, $\mathcal{B}_{S7}$ exactly simulates game G_{S7}. Thus,

$$|\Pr[S_{S6}] - \Pr[S_{S7}]| \leq \mathrm{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi}(\mathcal{B}_{S7}) \ .$$

Game G_{S8}

Difference from previous game: The game guesses a value $\ell \in [1, n_{\mathsf{kas}}]$, and aborts if the c_1 in the first C^* session to maliciously accept was not generated by the ℓth $\mathsf{Send}_{\mathsf{AS}}(C^*, S^*, \cdot)$ call.

Analysis of abort event: The guess is correct with probability at least $1/n_{\mathsf{kas}}$, thus

$$\Pr[S_{S7}] \leq n_{\mathsf{kas}} \Pr[S_{S8}] \ .$$

Game G_{S9}

Let (c_1^*, c_2^*) be the outputs of the ℓth $\mathsf{Send}_{\mathsf{AS}}(C^*, S^*, \cdot)$ call.

Difference from previous game: This game aborts if there exists any instance $\pi^j_{S^*}$ which does not reject when it receives a c_3 that was not output by a call to $\mathsf{Send}(C^*, i, (c_1^*, \cdot))$.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_3 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{S9}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{S8}, except as follows.

Reduction description: In $\mathsf{Send}(C^*, i, (c_1^*, \cdot))$, generate c_3 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger. (Note that this happens only once since it is preceded by a nonce check, and nonces are assumed to be unique by game G_{C1}.) Store c_3 and the corresponding plaintext in a list. In $\mathsf{Send}(S^*, j, (c_2^*, c_3))$ for any c_3, pass c_3 to the $\mathsf{INT\text{-}CTXT}$ challenger's Dec^* oracle; if c_3 was not in the aforementioned list, it rejects. Otherwise, it continues with corresponding plaintext. When it needs to generate c_4, it uses the Enc oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and stores c_4 and the corresponding plaintext in a (separate) list. In $\mathsf{Send}(C^*, i, c_4)$ for which $\pi^i_{C^*}.\mathsf{cid} = c_1^*$, use the plaintext corresponding to c_4 from the aforementioned list, or reject if none.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_3 in $\mathsf{Send}(S^*, \dots)$ is correct. Reveal queries: we cannot answer $\mathsf{Reveal}(C^*, i)$, but we don't need to: $\mathsf{F}(C^*, i)$ requires that there was no $\mathsf{Reveal}(C^*, i)$. We also cannot answer $\mathsf{Reveal}(S^*, j)$ for any j such that $\pi^j_{S^*}.\mathsf{cid} = c_2^*$, but we don't need to: $\mathsf{F}(S^*, i)$ requires there was no $\mathsf{Reveal}(S^*, j)$ for any j such that $(C^*, S^*, c_1^*, \pi^j_{S^*}.\mathsf{cid}) \in \mathsf{cids}$. $\mathsf{Corrupt}$ queries: no change compared to the previous game.

Winning: When the abort event occurs, some $\mathsf{Send}(S^*, j, (c_2^*, c_3))$ has received a c_3 that was not output by $\mathsf{Send}(C^*, i, (c_1^*, \cdot))$. Thus, none of $\mathcal{B}_{S9}$'s calls to its $\mathsf{INT\text{-}CTXT}$ Enc oracle ever returned c_3. But since this instance did not reject, it must be that $\mathcal{B}_{S9}$ call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c_3 did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{S9}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{S8}] - \Pr[S_{S9}]| \leq \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S9}) \ .$$

Game G_{S10}

Difference from previous game: This game aborts if some C^* instance accepts maliciously.

Analysis of abort event: We claim that if the abort event occurs, then the adversary has forged a valid c_4 ciphertext, breaking the integrity of the authenticated encryption scheme. More precisely, we build a reduction $\mathcal{B}_{S10}$ which interacts with an $\mathsf{INT\text{-}CTXT}$ challenger for Π and behaves exactly as in game G_{S9}, except as follows.

Reduction description: Let (c_1^*, c_2^*) be the outputs of the ℓth query to $\mathsf{Send}_{\mathsf{AS}}(C^*, \S^*, \cdot)$. In $\mathsf{Send}(C^*, i, (c_1^*, \cdot))$, generate c_3 by calling the $\mathrm{Enc}(\cdot)$ oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger. (Note that this happens only once since it is preceded by a nonce check, and nonces are assumed to be unique by game G_{C1}.) Store c_3 and the corresponding plaintext in a list. In $\mathsf{Send}(S^*, j, (c_2^*, c_3))$ for any c_3, pass c_3 to the $\mathsf{INT\text{-}CTXT}$ challenger's Dec^* oracle; if c_3 was not in the aforementioned list, it rejects. Otherwise, continue with the corresponding plaintext. When it needs to generate c_4, it uses the Enc oracle provided by the $\mathsf{INT\text{-}CTXT}$ challenger, and stores c_4 and the corresponding plaintext in a (separate) list. In $\mathsf{Send}(C^*, j, c_4)$ for which $\pi^i_{C^*}.\mathsf{cid} = c_1^*$, use the plaintext corresponding to c_4 from the aforementioned list, or reject if none.

Simulation correctness: Send queries: all messages are correctly distributed. Assuming the abort event does not occur, the simulation of processing of c_4 in $\mathsf{Send}(C^*, \dots)$ is correct. Reveal queries: we cannot answer $\mathsf{Reveal}(C^*, i)$ where $\pi^j_{C^*}.\mathsf{cid} = c_1^*$, but we don't need to: $\mathsf{F}(S^*, i)$ requires that there was no $\mathsf{Reveal}(C^*, i)$ for the target session i. We also cannot answer $\mathsf{Reveal}(S^*, j)$ where $\pi^j_{S^*}.\mathsf{cid} = c^*_2$, but we don't need to: $\mathsf{F}(C^*, i)$ requires there was no $\mathsf{Reveal}(S^*, j)$ for any j such that $(C^*, S^*, c^*_1, \pi^j_{S^*}.\mathsf{cid}) \in \mathsf{cids}$. $\mathsf{Corrupt}$ queries: no change compared to the previous game.

Winning: When the abort event occurs, some C^* instance $\pi^{i^*}_{C^*}$ has accepted maliciously. Note that the only place that the reduction makes calls to its $\mathsf{INT\text{-}CTXT}$ challenger's Enc oracle are either in calls to $\mathsf{Send}(C^*, i, (c^*_1, \cdot))$ (where Enc is called with plaintexts consisting of 2 components), or in $\mathsf{Send}(S^*, j, (c^*_2, \cdot))$ (where Enc is called with plaintexts consisting of 1 component). Since $\pi^{i^*}_{C^*}$ accepted maliciously, it did so with a session identifier $\pi^{i^*}_{C^*}.\mathsf{sid} = c^*_3 \| c^*_4$ where there is no matching session, namely, no j such that $\pi^j_{S^*}.\mathsf{sid} = c^*_3 \| c^*_4$. For every S^* instance that accepted with peer C^*, it did so with a sid that differed either in the c_3 component or the c_4 component. By the previous game, every S^* instance that accepted did so with a c_3 that was output by a call to $\mathsf{Send}(C^*, i, (c^*_1, \cdot))$. This means that S^*'s session identifier does not differ in the c_3 component, so it must differ in the c_4 component. Thus, no $\mathsf{Send}(S^*, j, (c^*_2, c^*_3))$ call ever output c^*_4. This is the only place that Enc oracle queries are made for plaintexts consisting of 1 component, so c^*_4 was not the output of an Enc query to the $\mathsf{INT\text{-}CTXT}$ challenger. But since this instance did not reject, it must be that $\mathcal{B}_{S10}$ call to its $\mathsf{INT\text{-}CTXT}$ Dec^* oracle with c^*_4 did not return $\perp$, which constitutes a forgery, and $\mathcal{B}_{S10}$ thereby wins its $\mathsf{INT\text{-}CTXT}$ game. Thus,

$$|\Pr[S_{S9}] - \Pr[S_{S10}]| \leq \mathrm{Adv}^{\mathsf{int\text{-}ctxt}}_{\Pi}(\mathcal{B}_{S10}) \ .$$

Analysis of Game G_{S10}

This game G_{S10} aborts if any client instance maliciously accepts, thus we have that

$$\Pr[S_{S10}] = 0$$

and our sequence of games for this case terminates. □

6 Session key security for Kerberos

A proof that Kerberos can be used to securely establish session keys faces the problem that standard authenticated key exchange security models cannot be used. Recall that classical AKE models (e.g. [BR94, CK01]) demand *session key indistinguishability*: in a "test" session, the adversary is challenged to distinguish the real session key from a random string of the same length. In Kerberos, the session key k_{CS} is already used to encrypt messages c_3 and c_4. Thus, the adversary can take the challenge, try to use it to decrypt c_3 and c_4, and thereby decide whether the challenge session key was real or random.

This problem is not unique to Kerberos. Many popular real-world cryptographic protocols have this characteristic, including TLS v1.0-1.2 and SSH v2. We have two options for dealing with this: we can try to argue indistinguishability of some intermediate key, or we can instead try to analyze the protocol as a *secure channel protocol* (which establishes and *uses* keys for authenticated encryption) rather than just a key establishment protocol. Early results on TLS focused on the first approach, but the first full analysis of TLS 1.2 by Jager et al. [JKSS12] was achieved via the introduction of the *authenticated and confidential channel establishment (ACCE)* security model, which addresses security of the channel resulting from the key exchange, rather than the indistinguishability of the key derived during the key exchange. We investigate the application of these two techniques to Kerberos.

For the first option, we recall that, as noted in Section 2, Kerberos can be run in an optional mode which establishes a separate "sub-session" key, k^*_{CS}. Unlike k_{CS}, this key is not used in the main

Kerberos exchange, so it is unaffected by the issues that prevented proofs of the indistinguishability of k_{CS} discussed at the beginning of this section. In Section 6.1, we show that the sub-session key is secure (indistinguishable from random) in an appropriate AKE security notion.

For the second option, unfortunately, neither the ACCE model nor the 3-party variant 3ACCE [BBF+17] can be applied to Kerberos. The ACCE and 3ACCE secure channel security notions specifically address the combination of a key exchange protocol and an authenticated encryption scheme. For TLS, this is fine: the TLS protocol specifies both a "handshake layer" (key exchange protocol) and subsequent "record layer" (authenticated encryption scheme). As noted earlier, Kerberos does not specify an exclusive record layer with which the Kerberos session key can only be used, and in fact does specify that it can be exported for arbitrary use [ZJH05]. Thus we cannot apply the ACCE or 3ACCE secure channel security notions since we have no specific encryption scheme with which the key will always be used.

This necessitates some generalization of the ACCE model. Krawczyk [Kra16] introduced such a generalization in the context of TLS 1.3 using *functional queries.* In Krawczyk's model, the classical Test query from AKE models is replaced by one or more functional queries (examples of which are the Encrypt and Decrypt queries in ACCE) on a black box containing the session key k, which may be adapted to each use case. We discuss this approach and how it can be applied to Kerberos in Section 6.2.

6.1 Indistinguishability of the sub-session key

Here we aim to show that the Kerberos sub-session key, denoted k^*_{CS} in Figure 2, is a secure session key. The typical security property for session keys dates from the original Bellare–Rogaway model [BR94]: indistinguishability of the session key from random, in an appropriate adversarial model. Figure 6 shows our security experiment 3-ake for security of the session key of a 3-party authenticated key exchange protocol Π. The experiment uses the same oracles as in Figure 3. It depends on a freshness predicate F (such as in Figure 4). The adversary is allowed to pick a single "test" session (using the Test oracle). The adversary is given either the real key from that session ($\pi^{i^*}_{A^*}.k$) or a random value chosen uniformly from the appropriate key space. The adversary wins if it can distinguish these with good probability:

$$\mathrm{Adv}^{\mathsf{3\text{-}ake}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{F}}(\mathcal{A}) = \left| \Pr\left[\mathrm{Exp}^{\mathsf{3\text{-}ake}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{F}}(\mathcal{A}) \Rightarrow \mathsf{true}\right] - \frac{1}{2} \right| .$$

The sub-session key can be shown to be secure (indistinguishable from random) in the sense of Figure 6. To be clear, in the formulation of this sub-session key mode of Kerberos, we say that session key output by an instance, $\pi^i_A.k$, is the sub-session key k^*_{CS}, rather than the session key k_{CS}. Assume that we are dealing with the sub-session key mode where the client picks the sub-session key and sends it in c_3. (The server-picked case sent in c_4 follows analogously.) The argument follows a similar structure to the proof of Theorem 1, with two similar cases and, most game hops in each case similar; only games G_{S9} and G_{C9} differ.

The sub-session key can be shown to be secure (indistinguishable from random) in the sense of Figure 6. To be clear, in the formulation of this sub-session key mode of Kerberos, we say that session key output by an instance, $\pi^i_A.k$, is the sub-session key k^*_{CS}, rather than the session key k_{CS}. Assume that we are dealing with the sub-session key mode where the client picks the sub-session key and sends it in c_3. (The server-picked case sent in c_4 follows analogously.) The argument follows a similar structure to the proof of Theorem 1, with two similar cases and, most game hops in each case similar; only games G_{S9} and G_{C9} differ. We now sketch each game hop.

In case S, we assume that the test session is at a client.

```
$\mathrm{Exp}^{\mathsf{3\text{-}ake}}_{\Pi,\mathcal{C},\mathcal{S},n_{\mathsf{sess}},\mathsf{F}}(\mathcal{A})$
1: // generate long-term keys
2: for $P \in \mathcal{C} \cup \mathcal{S}$ do
3:     $k_P \leftarrow_\$ \mathcal{K}$
4: $b \leftarrow_\$ \{0,1\}$
5: $A^* \leftarrow \perp$; $i^* \leftarrow \perp$
6: // run the experiment
7: $b' \leftarrow_\$ \mathcal{A}^{\mathsf{Send_{AS}},\mathsf{Send},\mathsf{Reveal},\mathsf{Corrupt},\mathsf{Test}}()$
8: // winning condition
9: return true iff $(b' = b) \wedge (\pi^i_A.\mathsf{status} = \mathsf{accepted})$
10:     $\wedge \neg\mathsf{corrupted}_{A^*} \wedge \neg\mathsf{corrupted}_{\pi^{i^*}_{A^*}.\mathsf{pid}} \wedge \mathsf{F}(A^*, i^*)$
```

```
$\mathsf{Test}(P, i)$:
1: if $P^* \neq \perp$ return $\perp$ // only 1 Test query allowed
2: if $\pi^{i^*}_{P^*}.\mathsf{status} \neq \mathsf{accepted}$ return $\perp$
3: $P^* \leftarrow P, i^* \leftarrow i$
4: if $b = 0$ then
5:     return $\pi^{i^*}_{P^*}.k$
6: else
7:     $k' \leftarrow_\$ \mathcal{K}^*$
8:     return $k'$
```

Figure 6: Session key security experiment for 3-party AKE protocol Π with freshness predicate F. Oracles $\mathsf{Send_{AS}}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{Corrupt}$ as defined in Figure 3.

- In game G_{S1}, we exclude nonce collision exactly as in Theorem 1. (Game hop based on birthday bound.)
- In game G_{S2}, we guess the client C^* to be the target of the test session. (Game hop based on guessing probability.)
- In game G_{S3}, we abort if C^* accepts any forged c_1 ciphertext, exactly as in Theorem 1. (Game hop based on INT-CTXT reduction.)
- In game G_{S4}, we replace all k_{CS} in c_1 ciphertexts to C^* with random keys, exactly as in Theorem 1. (Game hop based on IND-CPA reduction.)
- In game G_{S5}, we guess the peer S^* of the target of the test session. (Game hop based on guessing probability.)
- In game G_{S6}, we abort if S^* accepts any forged c_2 ciphertext, exactly as in Theorem 1. (Game hop based on INT-CTXT reduction.)
- In game G_{S7}, we replace all k_{CS} in c_2 ciphertexts to S^* with random keys, exactly as in Theorem 1. (Game hop based on IND-CPA reduction.)
- In game G_{S8}, we guess which KAS call generated the k_{CS} for the test session. (Game hop based on guessing probability.)
- In game G_{S9}, we replace the k^*_{CS} in c_3 with a random value. (Game hop based on IND-CPA reduction.)

Since the sub-session key k^*_{CS} in game G_{S9} is now completely independent of any transmitted messages, the adversary's advantage in game G_{S9} is 0.

In case C, we assume that the test session is at a server.

- In game G_{C1}, we exclude nonce collision exactly as in Theorem 1. (Game hop based on birthday bound.)
- In game G_{C2}, we guess the server S^* to be the target of the test session. (Game hop based on guessing probability.)
- In game G_{C3}, we abort if S^* accepts any forged c_2 ciphertext, exactly as in Theorem 1. (Game hop based on INT-CTXT reduction.)
- In game G_{C4}, we replace all k_{CS} in c_2 ciphertexts to S^* with random keys, exactly as in Theorem 1. (Game hop based on IND-CPA reduction.)

- In game G_{C5}, we guess the peer C^* of the target of the test session. (Game hop based on guessing probability.)
- In game G_{C6}, we abort if C^* accepts any forged c_1 ciphertext, exactly as in Theorem 1. (Game hop based on INT-CTXT reduction.)
- In game G_{C7}, we replace all k_{CS} in c_1 ciphertexts to C^* with random keys, exactly as in Theorem 1. (Game hop based on IND-CPA reduction.)
- In game G_{C8}, we guess which KAS call generated the k_{CS} for the test session. (Game hop based on guessing probability.)
- In game G_{C9}, we replace the k^*_{CS} in c_3 with a random value. (Game hop based on IND-CPA reduction.)

Since the sub-session key k^*_{CS} in game G_{C9} is now completely independent of any transmitted messages, the adversary's advantage in game G_{S9} is 0.

6.2 Security of (the hash of) the session key in other applications

We discussed already why we cannot prove it is safe to use session key directly in arbitrary applications. Krawczyk [Kra16] gave a generalization of the ACCE security model which allows one to prove that keys established in a session can be used in a specific setting, even when the adversary learns some information derived from the session key. In Krawczyk's framework, the setting in which the key is used is modelled by a functional test, and the information derived from the session key is modelled by a functional query. We now review this framework in more detail.

Let $F = \{f_p\}$ be a family of randomized functions, parameterized by p, and each taking a single argument k. We extend the security model of Section 3.1 with two "functional" queries associated to F:

- $\mathsf{FQuery}(P, i, p)$: The adversary provides a parameter p, and receives in return $f_p(\pi^i_P.k)$.
- $\mathsf{FTest}(P, i)$: The arguments P, i identify a *test session* π^i_P. FTest executes an arbitrary *functional test* interactively with the adversary. The challenger's input to the functional test is the test session's session key $\pi^P_A.k$, as well as a random bit b. The adversary outputs a bit b', and is said to win if $b' = b$ while maintaining the first four conditions in the predicate $\mathsf{malicious\text{-}accept}(P, i)$ (namely, that π^i_P has accepted, that P and its peer are not corrupted, and that the instance is fresh). Freshness is extended as follows: the adversary is permitted to issue FQuery queries to any instances that could also be revealed without violating freshness, and is also permitted to issue FQuery queries to the test session.

The functional test inside FTest must satisfy the condition that, if one replaces the input to the challenger with a random independent key, then the advantage of the adversary in winning the game is small; we call such a functional test *valid.*

Ideally, we would prove that Kerberos is secure in Krawczyk's framework for all function families F and all valid functional tests FTest. Unfortunately, we cannot: using the same key in an IND-CCA-like functional test that includes a decryption oracle (for the same authenticated encryption scheme Π as used in Kerberos) could allow for trial decryption of the Kerberos ciphertext (but not violate standard IND-CCA freshness since the functional test is not decrypting *its* challenge ciphertext). We know of no generic way to model when F is sufficiently different from Π to avoid this problem.

To achieve at least some positive result, we will consider the scenario where the Kerberos session key is first hashed before it is passed to the application; we call this protocol $H(\mathtt{3K})$. More specifically, the output session key is $H(k)$ (but c_3 and c_4 continue to be encrypted with k directly). Any applications that take the Kerberos session key and run it through a hash function, key derivation function, or pseudorandom function before using it fit this pattern.

Our goal will be to prove that $H(\mathsf{3K})$ is secure in Krawczyk's framework for all functional families F and all valid functional tests FTest. We will do so by introducing a joint assumption on the hash function H and the authenticated encryption scheme Π which is an extension of Fischlin's definition of a *key-hiding symmetric encryption scheme* [Fis99, Def. 6]. Our joint assumption intuitively says that encryptions under the key are independent of the hash of the key.

Fischlin's definition of a key-hiding symmetric encryption scheme is as follows. Let D be a distinguishing algorithm. For a symmetric encryption scheme Π with key space $\mathcal{K}$, define

$$\begin{aligned}\mathrm{Adv}^{\mathsf{kh}}_{\Pi}(D) = \Big| &\Pr\left[k \leftarrow_\$ \mathcal{K}; D^{\mathrm{Enc}_k(\cdot),\mathrm{Enc}_k(\cdot)}() \Rightarrow 1\right] \\ &- \Pr\left[k, k' \leftarrow_\$ \mathcal{K}; D^{\mathrm{Enc}_k(\cdot),\mathrm{Enc}_{k'}(\cdot)}() \Rightarrow 1\right]\Big| .\end{aligned}$$

In other words, in a key-hiding symmetric encryption scheme, the adversary cannot tell whether ciphertexts are being generated using the same key or different keys.

Based on Fischlin's definition of a key-hiding symmetric encryption scheme, we define *H-key-hiding symmetric encryption scheme* as follows. Let D be a distinguishing algorithm, Π be a symmetric encryption scheme Π with key space $\mathcal{K}$, and $H : \mathcal{K} \to \mathcal{K}'$ be a hash function. Define

$$\begin{aligned}\mathrm{Adv}^{\mathsf{hkh}}_{\Pi,H}(D) = \Big| &\Pr\left[k \leftarrow_\$ \mathcal{K}; D^{\mathrm{Enc}_k(\cdot)}(H(k)) \Rightarrow 1\right] \\ &- \Pr\left[k, k' \leftarrow_\$ \mathcal{K}; D^{\mathrm{Enc}_k(\cdot)}(H(k')) \Rightarrow 1\right]\Big| .\end{aligned}$$

In other words, in an H-key-hiding symmetric encryption scheme, the adversary cannot tell whether it is being given the hash of the encryption key, or the hash of different key, even with access to an encryption oracle.

First, we show that the use of an H-key-hiding symmetric encryption scheme in Kerberos is secure in Krawczyk's framework. Second, we give some justification that this new assumption is reasonable by showing that it holds in the random oracle model, assuming the encryption scheme is secure against key recovery under chosen plaintext attacks.

Theorem 2 (Security of $H(\mathsf{3K})$ in Krawczyk's framework)**.** *If H is a random oracle and, in addition to all of the assumptions in Theorem 1, we also have that Π is an H-key-hiding symmetric encryption scheme, then $H(\mathsf{3K})$, is secure in Krawczyk's framework for any functional family F and any valid functional test FTest.*

Proof sketch. We first apply the same sequence of games in the proof of Theorem 1 to $H(\mathsf{3K})$, except everywhere we are considering the test session rather than the first session to maliciously accept. By the end of this sequence of games, we have guessed participants C^* and S^*, and know that the test session involves c_1^* or c_2^* generated by the KAS.

In the next game, we replace the session key $H(k)$ with $H(r)$ for $r \leftarrow_\$ \mathcal{K}$ in every C^* or S^* session involving c_1^* or c_2^*. We don't have to worry about inconsistency of Reveal queries since those are prohibited by freshness. Indistinguishability of this game from the previous game follows under H-key-hiding of Π, introducing a $\mathrm{Adv}^{\mathsf{hkh}}_{\Pi,H}(D)$ term in the advantage.

This game is our final game: the session key of the test session are indistinguishable from random since the input to the random oracle model is independent and unknown to the adversary. This allows application of Krawczyk's framework: the functional test being valid thereby implies that the advantage of the adversary winning is small. □

We have two questions to answer: Does this approach say anything meaningful about uses of Kerberos? And is the H-key-hiding notion reasonable?

Utility of the approach can be seen by looking at RFCs 3961 [Rae05b] and 4121 [ZJH05], which both discuss use of Kerberos session keys to derive subsequent keys; their mechanism is in effect "hashing" the session key using a block cipher (e.g. RFC 3961 [Rae05b] Section 5.1).

Finally: is the H-key-hiding notion reasonable? We can provide a heuristic justification of it in the random oracle model, under the assumption the symmetric encryption is secure against key recovery under chosen plaintext attack (KR-CPA), which of course is implied by the standard IND-CPA property. (In the HKH experiment in the random oracle model, D also gets access to the random oracle.)

Theorem 3 (KR-CPA $\implies$ H-key-hiding in ROM)**.** *If Π is a symmetric encryption scheme secure against key recovery under chosen plaintext attacks (KR-CPA) and H is a random oracle, then Π is an H-key-hiding symmetric encryption scheme. More precisely, if D is an algorithm that makes q queries to its H oracle, then there exists an algorithm R, described in the proof, such that*

$$\mathrm{Adv}_{\Pi}^{\mathsf{hkh}}(D) \leq q \cdot \mathrm{Adv}_{\Pi}^{\mathsf{kr\text{-}cpa}}(R) \ ,$$

where R has approximately the same runtime as D.

The definition of KR-CPA security appears in Appendix C.

Proof. Suppose D is an adversary against the H-key-hiding security of Π. Since H is a random oracle, D has no advantage unless it queries H on k. We construct a reduction R against the KR-CPA security of R. R interacts with a KR-CPA challenger. When D makes an Enc query, R passes that query directly to its challenger and returns the result. When D makes an H query, R simulates a random oracle, maintaining a list of queries. When D terminates, R picks one of the q queries at random from the list of queries and outputs it as its guess of the key. When D succeeds in distinguishing, k appears on R's list so R succeeds with probability $1/q$. □

Of course we would prefer to have a justification of H-key-hiding in the standard model (e.g., assuming H is a pseudorandom generator rather than a random oracle). This is tricky: one could construct degenerate encryption schemes Enc and functions H that are individually secure but together interfere with each other, especially if Enc and H are built from common building primitives. Additionally, Fischlin [Fis99], in his discussion about key-hiding being non-trivial, notes an example in which the Hamming weight is leaked, and this extends in the obvious was to H-key-hiding as well. In practice, many natural constructions likely would not interfere with each other, but in a proof we would need a formalization of "similarity" of algorithm structure, which is outside the scope of this effort.

7 Discussion

Since this work gives provable security results with concrete bounds, it can be used to pick parameters in a protocol instantiation. Here a key issue is "tightness": the smaller the multiplicative factor between the advantage of breaking the protocol and the advantage of breaking the underyling primitives, the "tighter" the result, and therefore the smaller the overhead that the proof incurs on the parameter sizes. Looking at Theorem 1, we see that the multiplicative factors are n_{clnts} and

n_{srvrs}, which are the number of clients and servers, and n_{kas}, the number of queries to the Kerberos authentication server. In practice n_{clnts} and n_{srvrs} is likely to be smaller relative to the security parameter (e.g., on the order of 2^{10} or 2^{20} compared to 2^{128}), so they do not impose a huge penalty on tightness. The number of KAS queries—roughly the number of sessions—could be much larger over a long deployment. One solution is to apply rate-limiting on the KAS to ensure this number does not grow too large. Alternatively, one could instantiate the authenticated encryption scheme with higher security for the message authentication component. Noting as well the birthday bound involving $n_{\mathsf{clnts}} \cdot n_{\mathsf{sess}}$ versus the size of the nonce, this suggests one should use sufficiently large nonces, e.g. 256-bit. If a weak cipher is used in Kerberos—such as DES with 56-bit keys and a lower security checksum [Rae05b]—then the multiplicative factors may overwhelm the advantage yielding no security assurance; but with high security ciphers with larger keys, our theorem yields a meaningful security bound.

Another benefit of our explicit reductionist proof is that we see the structure of the proof in the computational setting, and so we can confirm that Song's quantum lifting lemma [Son14] can be applied to yield security against quantum adversaries (assuming quantum security of the underlying authenticated encryption scheme). While the symbolic (Dolev–Yao) proof for Kerberos of Backes et al. [BCJ+06a, BCJ+11] can be lifted by results of BPW [BPW03] to the computational setting for classical adversaries, we do not know the structure of the implicit computational proof to know whether Song's quantum lemma can be applied for quantum adversaries.

Finally, by providing results about the security of the sub-session key as well as the hash of the session key, we show that it safe to use these keys in subsequent cryptographic algorithms like authenticated encryption. Although it would be nice to be able to prove results about the main session key, we cannot do so due to its use directly in the protocol itself, as we explained above, and thus can only achieve provable security results in these alternates, the first of which (sub-session keys) is standardized and the second of which (hashing the session key) is a straightforward variant that would be simple for practitioners to use.

Acknowledgements

We acknowledge helpful discussions on earlier versions of this work with Florian Bergsma, Choudary Gorantla, and Florian Kohlar. D.S. is supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2016-05146 and an NSERC Discovery Accelerator Supplement grant.

References

[ABV+04] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFCs 5247, 7057.

[AHSM10] N. T. Abdelmajid, M. Alamgir Hossain, S. Shepherd, and K. Mahmoud. Improved kerberos security protocol evaluation using modified BAN logic. In *10th IEEE International Conference on Computer and Information Technology, CIT 2010, Bradford, West Yorkshire, UK, June 29-July 1, 2010*, pages 1610–1615, 2010.

[Bac04] Michael Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS 2004*, volume 3193 of *LNCS*, pages 89–108. Springer, Heidelberg, September 2004.

[BAN90] Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[BBF$^+$17] Karthikeyan Bhargavan, Ioana Boureanu, Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. Content delivery over TLS: a cryptographic analysis of keyless SSL. In *2017 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, April 2017.

[BCJ$^+$06a] Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Cryptographically sound security proofs for basic and public-key kerberos. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS 2006*, volume 4189 of *LNCS*, pages 362–383. Springer, Heidelberg, September 2006.

[BCJ$^+$06b] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Formal analysis of kerberos 5. *Theor. Comput. Sci.*, 367(1-2):57–87, 2006.

[BCJ$^+$11] Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Cryptographically sound security proofs for basic and public-key kerberos. *Int. J. Inf. Sec.*, 10(2):107–134, 2011.

[BCJS02] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. A formal analysis of some properties of kerberos 5 using MSR. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, page 175, 2002.

[BCJS03] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. Verifying confidentiality and authentication in kerberos 5. In *Software Security - Theories and Systems, Second Mext-NSF-JSPS International Symposium, ISSS 2003, Tokyo, Japan, November 4-6, 2003, Revised Papers*, pages 1–24, 2003.

[BD95] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system (extended abstract). In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 275–286. Springer, Heidelberg, May 1995.

[BF09] Manuel Barbosa and Pooya Farshim. Security analysis of standard authentication and key agreement protocols utilising timestamps. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 235–253. Springer, Heidelberg, June 2009.

[BJ17] Chris Brzuska and Håkon Jacobsen. A modular security analysis of EAP and IEEE 802.11. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 335–365. Springer, 2017.

[BJST08] Buno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08*, pages 87–99. ACM Press, March 2008.

[BK11] Alexandra Boldyreva and Virendra Kumar. Provable-security analysis of authenticated encryption in kerberos. *IET Information Security*, 5(4):207–219, 2011.

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

[BM91] Steven M. Bellovin and Michael Merritt. Limitations of the kerberos authentication system. In *Proceedings of the Usenix Winter 1991 Conference, Dallas, TX, USA, January 1991*, pages 253–268, 1991.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008.

[BP98a] Giampaolo Bella and Lawrence C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, and Dieter Gollmann, editors, *ESORICS'98*, volume 1485 of *LNCS*, pages 361–375. Springer, Heidelberg, September 1998.

[BP98b] Giampaolo Bella and Lawrence C. Paulson. Mechanising BAN kerberos by the inductive method. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, pages 416–427, 1998.

[BP03] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the needham-schroeder-lowe public-key protocol. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, pages 1–12, 2003.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 220–230. ACM Press, October 2003.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th ACM STOC*, pages 57–66. ACM Press, May / June 1995.

[BR97] Giampaolo Bella and Elvinia Riccobene. Formal analysis of the kerberos authentication system. *J. UCS*, 3(12):1337–1381, 1997.

[CJS$^+$08] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key kerberos. *Inf. Comput.*, 206(2-4):402–424, 2008.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

[DDM$^+$05] Anupam Datta, Ante Derek, John C. Mitchell, Vitaly Shmatikov, and Mathieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic (invited lecture). In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 16–29. Springer, Heidelberg, July 2005.

[DDMP03] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system for security protocols and its logical formalization. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003), 30 June - 2 July 2003, Pacific Grove, CA, USA*, pages 109–125. IEEE Computer Society, 2003.

[DLS97] Bryn Dole, Steven W. Lodin, and Eugene H. Spafford. Misplaced trust: Kerberos 4 session keys. In *NDSS'97*. IEEE Computer Society, February 1997.

[DSZ16] Benjamin Dowling, Douglas Stebila, and Greg Zaverucha. Authenticated network time synchronization. In *Proc. 25th USENIX Security Symposium 2016*. USENIX, August 2016.

[EBS95] James T. Ellis, David M. Balenson, and Robert W. Shirey, editors. *1995 Symposium on Network and Distributed System Security, (S)NDSS '95, San Diego, California, February 16-17, 1995*. IEEE Computer Society, 1995.

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1193–1204. ACM Press, November 2014.

[Fis99] Marc Fischlin. Pseudorandom function tribe ensembles based on one-way permutations: Improvements and applications. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 432–445. Springer, Heidelberg, May 1999.

[FLW09] Kai Fan, Hui Li, and Yue Wang. Security analysis of the kerberos protocol using BAN logic. In *Proceedings of the Fifth International Conference on Information Assurance and Security, IAS 2009, Xi'An, China, 18-20 August 2009*, pages 467–470, 2009.

[Gan95] Ravi Ganesan. Yaksha: augmenting kerberos with public key cryptography. In Ellis et al. [EBS95], pages 132–143.

[Har12] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), October 2012.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Heidelberg, August 2012.

[KEGM14] Yoney Kirsal-Ever, Agozie Eneh, Orhan Gemikonakli, and Leonardo Mostarda. Analysing the combined kerberos timed authentication protocol and frequent key renewal using CSP and rank functions. *TIIS*, 8(12):4604–4623, 2014.

[KLLN16] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 207–237. Springer, Heidelberg, August 2016.

[KN93] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510 (Historic), September 1993. Obsoleted by RFCs 4120, 6649.

[Koh90] John T. Kohl. The use of encryption in Kerberos for network authentication. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 35–43. Springer, Heidelberg, August 1990.

[Kra16] Hugo Krawczyk. A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1438–1450. ACM Press, October 2016.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[LP10] Yongjian Li and Jun Pang. Extending the strand space method with timestamps: Part II application to kerberos V. *J. Information Security*, 1(2):56–67, 2010.

[LYZZ09] Qin Li, Fan Yang, Huibiao Zhu, and Longfei Zhu. Formal modeling and analyzing kerberos protocol. In *CSIE 2009, 2009 WRI World Congress on Computer Science and Information Engineering, March 31 - April 2, 2009, Los Angeles, California, USA, 7 Volumes*, pages 813–819, 2009.

[McM95] P. V. McMahon. SESAME V2 public key and authorisation extensions to kerberos. In Ellis et al. [EBS95], pages 114–131.

[Mea96] Catherine Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *ESORICS'96*, volume 1146 of *LNCS*, pages 351–364. Springer, Heidelberg, September 1996.

[Met15] Sean Metcalf. Red vs. blue: Modern active directory - attacks, detection, and protection. `https://www.blackhat.com/docs/us-15/materials/us-15-Metcalf-Red-Vs-Blue-Modern-Active-Directory-Attacks-Detection-And-Protection-wp.pdf`, 2015.

[NS78a] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[NS78b] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the Association for Computing Machinery*, 21(21):993–999, December 1978.

[NYHR05] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806, 7751.

[OR87] David J. Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[Rae05a] K. Raeburn. Advanced Encryption Standard (AES) Encryption for Kerberos 5. RFC 3962 (Proposed Standard), February 2005.

[Rae05b] K. Raeburn. Encryption and Checksum Specifications for Kerberos 5. RFC 3961 (Proposed Standard), February 2005.

[RDDM07] Arnab Roy, Anupam Datta, Ante Derek, and John C. Mitchell. Inductive proofs of computational secrecy. In Joachim Biskup and Javier López, editors, *ESORICS 2007*, volume 4734 of *LNCS*, pages 219–234. Springer, Heidelberg, September 2007.

[RDM07] Arnab Roy, Anupam Datta, and John C. Mitchell. Formal proofs of cryptographic security of diffie-hellman-based protocols. In Gilles Barthe and Cédric Fournet, editors, *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, volume 4912 of *Lecture Notes in Computer Science*, pages 312–329. Springer, 2007.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.

[SB12] Christoph Sprenger and David Basin. Refining key establishment. In *CSF 2012*, pages 230–246. IEEE, 2012.

[SC97] Marvin A. Sirbu and John C.-I. Chuang. Distributed authentication in Kerberos using public key cryptography. In *NDSS'97*. IEEE Computer Society, February 1997.

[Sch14] Jörg Schwenk. Modelling time for authenticated key exchange protocols. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 277–294. Springer, Heidelberg, September 2014.

[Sch16] Jörg Schwenk. Nonce-based kerberos is a secure delegated AKE protocol. Cryptology ePrint Archive, Report 2016/219, 2016. `http://eprint.iacr.org/2016/219`.

[SNS88] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter Conference. Dallas, Texas, USA, January 1988*, pages 191–202, 1988.

[Son14] Fang Song. A note on quantum security for post-quantum cryptography. In Michele Mosca, editor, *PQCrypto 2014*, volume 8772 of *LNCS*, pages 246–265. Springer, 2014.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In Li Gong and Jacques Stern, editors, *ACM CCS 96*, pages 31–37. ACM Press, March 1996.

[STW00] Michael Steiner, Gene Tsudik, and Michael Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, August 2000.

[WGC12] Kazunori Wagatsuma, Yuichi Goto, and Jingde Cheng. Formal analysis of cryptographic protocols by reasoning based on deontic relevant logic: A case study in needham-schroeder shared-key protocol. In *International Conference on Machine Learning and Cybernetics, ICMLC 2012, Xian, Shaanxi, China, July 15-17, 2012, Proceedings*, pages 1866–1871, 2012.

[YHR04] Tom Yu, Sam Hartman, and Kenneth Raeburn. The perils of unauthenticated encryption: Kerberos version 4. In *NDSS 2004*. The Internet Society, February 2004.

[ZJH05] L. Zhu, K. Jaganathan, and S. Hartman. The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2. RFC 4121 (Proposed Standard), July 2005. Updated by RFCs 6112, 6542, 6649.

[ZT06] L. Zhu and B. Tung. Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). RFC 4556 (Proposed Standard), June 2006. Updated by RFC 6112.

A Attacks on Kerberos

In this appendix we mention two types of attacks that Kerberos is *not* designed to protect against. These inform the boundaries of any security definition for Kerberos.

Replay attacks. Any network adversary can simply replay messages (c_2, c_3) to the same server S as long as $\text{timeok}(t_{KAS}, t_C; \text{now}())$ is true. This attack would make another server instance at server S accept, but would not lead to the establishment of a secure channel, since any client instance at client C would choose a fresh, increased timestamp t'_C for each new session, and would reject the old t_C.

Such replay attacks could completely be eliminated by having the server maintain a persistent variable t_C^{last} at the server, which would be set to the client's supplied t_C on the first reception from C of a c_3 message. Each subsequent message would have its t_C checked against t_C^{last}; the server rejecting if the timestamps are out of order, otherwise accepting and updating t_C^{last} with the new t_C.

Transcript modification attacks. In this attack, depicted in Figure 7, an active network adversary may change the transcript of both client and server, but both parties nevertheless accept. Specifically the adversary replaces the c_2 delivered to the client with a fake value, but then delivers the original c_2 to the server. Neither party will detect that the adversary has performed this. This attack does not break any security goals of Kerberos, but shows that we cannot use the whole transcript to define partnering.

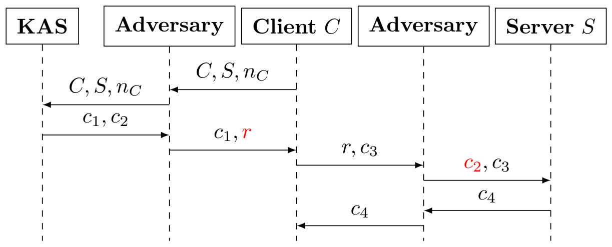

Figure 7: Transcript modification attack on Kerberos.

B Modelling timestamps

Kerberos uses timestamps to protect servers from replay attacks. Since we are not modelling such attacks, and in order to keep our proof generic, we did not explicitly specify the functions now() and timeok(). They can be instantiated with different functions, as long as the following requirements are met:

- *Approximating time using* now(). Ideally, one would instantiate the function now() with physical time, i.e. with perfectly synchronized local clocks. However, our formal model is based on Turing machines, which model only computation steps, not physical processes. For Kerberos protocol to be functional in the presence of a benign adversary, it suffices that now() is a local function at each party that returns monotonically increasing values.
- *Checking freshness using* timeok(). The timeok function is a local function of a server which takes as input two received timestamps t_{KAS} and t_C, and a local timestamp $t \leftarrow \text{now}()$, and

returns true or false. Typically it checks if the difference between the fresh local time t and each of the two received timestamps is below a defined threshold.

Few reduction-based security proofs consider time. Barbosa and Farshim [BF09] introduced a model in which each party maintains a local clock, and the adversary can control each party's flow of time by calling a tick query at each party. In this model, the adversary controls all clocks, and even a benign adversary must be activate to ensure normal protocol operation. The adversary can arbitrarily desynchronize all clocks. Barbosa and Farshim used this approach to model timed authenticated key exchange protocols; Dowling et al. [DSZ16] use a similar approach for authenticated network time synchronization (i.e., simulating a global clock using local clocks). Schwenk [Sch14] models time as a global counter $\mathcal{T}$ which is increased on each call of the function. A network adversary may manipulate the responses to global time queries, but since parties keep a local counter, an attack will be detected if the manipulated value is smaller than the local counter.

One of the results of our security analysis of Kerberos is that clients do not rely on timestamps at all. After decrypting c_1, the client may safely ignore the timestamp t_{KAS}, since freshness is guaranteed through the nonce n_C. In message c_4, the timestamp t_C is a "predictable nonce" and does not play a role at all in the security proof – due to the properties of the authenticated encryption scheme, it would be sufficient to simply encrypt a constant value here.

C Encryption definitions

A *symmetric encryption scheme* Π consists of the following two algorithms, along with an associated key space $\mathcal{K}$ and message space $\mathcal{M}$:

- $\mathrm{Enc}(k, m) \xrightarrow{\$} c$: A probabilistic encryption algorithm that takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext c. Sometimes denoted $\mathrm{Enc}_k(m)$.
- $\mathrm{Dec}(k, c) \to m$ or $\perp$: A deterministic decryption algorithm that takes as input a key $k \in \mathcal{K}$ and a ciphertext c, and outputs a message m or a distinguished error symbol $\perp \notin \mathcal{M}$. Sometimes denoted $\mathrm{Dec}_k(c)$.

The scheme is said to be *correct* if, for all $m \in \mathcal{M}$,

$$\Pr\left[k \leftarrow_{\$} \mathcal{K}; \mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m\right] = 1 \ .$$

To be an *authenticated encryption* scheme, a symmetric encryption scheme should satisfy two main security properties: confidentiality and integrity [BN08]. Confidentiality is modelled as indistinguishability under either chosen plaintext or chosen ciphertext attacks using the "left or right" notation, shown in Figure 8. Integrity can be of either plaintexts or ciphertexts, shown in Figure 9. The difference between the two integrity notions is subtle, but important: if an authenticated encryption scheme is only INT-PTXT-secure, the adversary may be able to change the ciphertext, as long as the decrypted plaintext remains the same. (Some workd prefer to use combined confidentiality+integrity ("all-in-one") definitions [RS06]. We stick with separate notions, since our proofs will separately use either integrity or confidentiality at different stages, and it is instructive to see which property is in use.)

We also recall a weaker notion, called *key recovery under chosen plaintext attacks* (KR-CPA), which suffices for some of our proofs, and which is clearly implied by IND-CPA.

```
$\mathrm{Exp}_{\Pi}^{\textsf{kr-cpa}}(\mathcal{A})$
1: $k \leftarrow_{\$} \mathcal{K}$
2: $k' \leftarrow_{\$} \mathcal{A}^{\mathrm{Enc}_k(\cdot)}()$
3: if $k = k'$ return 1 else return 0
```

```
$\mathrm{Exp}_{\Pi}^{\textsf{ind-cpa}}(\mathcal{A})$
1: $k \leftarrow_{\$} \mathcal{K}$
2: $b \leftarrow_{\$} \{0,1\}$
3: $b' \leftarrow_{\$} \mathcal{A}^{\mathrm{Enc}_k(\mathrm{LR}_b(\cdot,\cdot))}()$
4: if $b = b'$ return 1 else return 0

$\mathrm{LR}_b(m_0, m_1)$
1: return $m_b$
```

```
$\mathrm{Exp}_{\Pi}^{\textsf{int-ctxt}}(\mathcal{A})$
1: $k \leftarrow_{\$} \mathcal{K}$
2: $b \leftarrow_{\$} \{0,1\}$
3: $b' \leftarrow_{\$} \mathcal{A}^{\mathrm{Enc}_k(\mathrm{LR}_b(\cdot,\cdot)),\mathrm{Dec}_k(\cdot)}()$
4: if $\mathcal{A}$ queries an output of its Enc oracle to its Dec oracle then
5:     return $x \leftarrow_{\$} \{0,1\}$
6: if $b = b'$ return 1 else return 0
```

Figure 8: Security experiments for key recovery under chosen plaintext attack and indistinguishability under chosen plaintext and chosen ciphertext attack of a symmetric encryption scheme Π against an adversary $\mathcal{A}$.

```
$\mathrm{Exp}_{\Pi}^{\textsf{int-ptxt}}(\mathcal{A})$
1: $k \leftarrow_{\$} \mathcal{K}$
2: run $\mathcal{A}^{\mathrm{Enc}_k(\cdot),\mathsf{Dec}^*(\cdot)}()$
3: if $\mathcal{A}$ makes a query $c$ to $\mathsf{Dec}^*$ such that $\mathsf{Dec}^*$ returned 1 and $m = \mathrm{Dec}_k(c)$ was never a query to Enc then
4:     return 1
5: else
6:     return 0
```

```
$\mathrm{Exp}_{\Pi}^{\textsf{int-ctxt}}(\mathcal{A})$
1: $k \leftarrow_{\$} \mathcal{K}$
2: run $\mathcal{A}^{\mathrm{Enc}_k(\cdot),\mathsf{Dec}^*(\cdot)}()$
3: if $\mathcal{A}$ makes a query $c$ to $\mathsf{Dec}^*$ such that $\mathsf{Dec}^*$ returned 1 and $c$ was never a response of Enc then
4:     return 1
5: else
6:     return 0
```

```
$\mathsf{Dec}^*(c)$
1: if $\mathrm{Dec}_k(c) = \perp$ then return 0 else return 1
```

Figure 9: Security experiments for plaintext- and ciphertext-integrity of an authenticated encryption scheme Π against an adversary $\mathcal{A}$.

We define corresponding advantages for each experiment:

$$\begin{aligned}
\mathrm{Adv}_{\Pi}^{\textsf{kr-cpa}}(\mathcal{A}) &= \Pr[\mathrm{Exp}_{\Pi}^{\textsf{kr-cpa}}(\mathcal{A}) \Rightarrow 1] \\
\mathrm{Adv}_{\Pi}^{\textsf{ind-cpa}}(\mathcal{A}) &= \left|\Pr[\mathrm{Exp}_{\Pi}^{\textsf{ind-cpa}}(\mathcal{A}) \Rightarrow 1] - 1/2\right| \\
\mathrm{Adv}_{\Pi}^{\textsf{ind-cca}}(\mathcal{A}) &= \left|\Pr[\mathrm{Exp}_{\Pi}^{\textsf{ind-cca}}(\mathcal{A}) \Rightarrow 1] - 1/2\right| \\
\mathrm{Adv}_{\Pi}^{\textsf{int-ptxt}}(\mathcal{A}) &= \Pr[\mathrm{Exp}_{\Pi}^{\textsf{int-ptxt}}(\mathcal{A}) \Rightarrow 1] \\
\mathrm{Adv}_{\Pi}^{\textsf{int-ctxt}}(\mathcal{A}) &= \Pr[\mathrm{Exp}_{\Pi}^{\textsf{int-ctxt}}(\mathcal{A}) \Rightarrow 1]
\end{aligned}$$