Nonce-based Kerberos is a Secure Delegated AKE Protocol

Jörg Schwenk
joerg.schwenk@rub.de
Horst Görtz Institute for IT Security
Ruhr University Bochum

Abstract. Kerberos is one of the most important cryptographic protocols, first because it is the basic authentication protocol in Microsoft’s Active Directory and shipped with every major operating system, and second because it served as a model for all Single-Sign-On protocols (e.g. SAML, OpenID, MS Cardspace, OpenID Connect). Its security has been confirmed with several Dolev-Yao style proofs [1–12], and attacks on certain versions of the protocol have been described [13, 14].

However despite its importance, despite its longevity, and despite the wealth of Dolev-Yao-style security proofs, no reduction based security proof has been published until now. This has two reasons: (1) All widely accepted formal models either deal with two-party protocols, or group key agreement protocols (where all entities have the same role), but not with 3-party protocols where each party has a different role. (2) Kerberos uses timestamps and nonces, and formal security models for timestamps are not well understood up to now.

As a step towards a full security proof of Kerberos, we target problem (1) here: We propose a variant of the Kerberos protocol, where nonces are used instead of timestamps. This requires one additional protocol message, but enables a proof in the standard Bellare-Rogaway (BR) model. The key setup and the roles of the different parties are identical to the original Kerberos protocol.

For our proof, we only require that the authenticated encryption and the message authentication code (MAC) schemes are secure. Under these assumptions we show that the probability that a client or server process oracle accepts maliciously, and the advantage of an adversary trying to distinguish a real Kerberos session key from a random value, are both negligible.

One main idea in the proof is to model the Kerberos server as a public oracle, so that we do not have to consider the security of the connection client–Kerberos. This idea is only applicable to the communication pattern adapted by Kerberos, and not to other 3-party patterns (e.g. EAP protocols).
1 Introduction

1.1 Reduction based vs. Doley-Yao security proofs

Reduction based security proofs are the basis of modern cryptography: New complex cryptographic constructions are shown to be secure by giving a polynomial reduction to a (small) set of well-established security assumptions (e.g. the factoring assumption or the decisional Diffie-Hellman assumption). Reduction based proofs for cryptographic protocols (following [15]) mostly deal with either (a) two-party protocols where each party has a different role (i.e. it executes a different code), or (b) with group key agreement protocols where all parties have the same role (i.e. they all execute the same code). Examples are (a) SSL/TLS [16], SSH [17] and HMQV [18], and (b) Burmester-Desmedt [19] and [20,21].

For Kerberos-type, three-party protocols where each part has a different role, publications on reduction-based security are rare, although they also play an important role in practical IT security: EAP protocols (mobile device – access point – RADIUS server) are used to secure enterprise WLAN environments, Kerberos is the basis for MS Active Directory security, and novel Single-Sign-On (SSO, identity provider – browser – relying party) protocols like SAML and OpenID Connect strive to replace password-based authentication on the Internet. One of the sparse results published is the paper of Bellare et al. [22], which describes a reduction-based model for communication pattern 3 in Figure 1.

A second (earlier) important line of research was started with the seminal paper of Burrows, Abadi and Needham [23] (the latter being the inventor of a predecessor of Kerberos, together with Michael D. Schroeder): Abstractions of (idealized) cryptographic operations were used for logic reasoning, e.g. that equations like $D_k(E_k(m)) = m$ always hold. Here research on Kerberos and related protocols has a long history, with many important results, which will be sketched in the Related Work section.

Serious efforts have been made to unite these two approaches (e.g. [24]), e.g. by adapting the cryptographic primitives used to the requirements of Dolev-Yao proofs, but please note that these approaches still use a methodology different from reduction-based proofs.

1.2 Three-party communication patterns

The different communication patterns used in three-party scenarios are depicted in Figure 1. The key setup in all these patterns is the same: Parties A and B each share a symmetric key with the trusted party TP. The goal is always to establish an authenticated symmetric session key between A and B, by using only symmetric cryptographic primitives.

In the early days of cryptographic protocols, different authenticated key establishment (AKE) protocols using only symmetric cryptography have been proposed: Needham-Schroeder [25] for Pattern 1, Otway-Rees [26, 27] for Pattern
2, and the Wide Mouthed Frog protocol [23] for Pattern 3. Their goal was to solve the symmetric key distribution problem: If \( n \) parties want to communicate securely using symmetric cryptography, a symmetric key has to be established between each of the \( n(n - 1) \) pairs of parties. Instead of \( n(n - 1) \) keys, only \( n \) keys have to be manually configured in the setup phase, and all other keys are established automatically.

Although it seemed that with the advent of Public Key Infrastructures (PKI) this type of key management was outdated, recent discussions on quantum computers may revive the interest in this topic: while breaking all public key algorithms used today, quantum algorithms affect symmetric algorithms only slightly, by requiring a doubling of the key length. Thus e.g. Kerberos with 256 Bit keys will still be secure in a post-quantum world.

Since the three example protocols mentioned above were proposed by pioneers of logical protocol analysis, there are many Dolev-Yao-style analysis papers, e.g. [3, 5, 6, 23, 28–33]. However, the only reduction-based security model was proposed for the least used of the three patterns, namely for Pattern 3 [22].

1.3 Kerberos

Kerberos was developed to protect network services in the MIT project Athena. It was inspired by the Needham-Schroeder protocol [34], and its primary designers were Steve Miller and Clifford Neuman.

Version 4 was published in [35], and Version 5 in RFC 1510 [36] in 1993. The current version of the protocol is described in RFC 4120 [37]. Version 5 eliminated some security problems with the initial protocol, and was subsequently adopted in Microsoft Windows 2000 and all later OS versions as the default authentication protocol. Many other operating systems like FreeBSD, Apple’s Mac OS X and Red Hat Enterprise Linux, also support Kerberos.

---

1 Please note that this security model cannot simply be reused for Pattern 1 and 2, because it it heavily depends in Pattern 3.
The body of the Kerberos protocol consists of two or three 2-message exchanges. The two-exchange case is the basis for the version of the nonce-based Kerberos variant analyzed in this paper, and we will discuss the extension to the three-exchange case in Section 6.

The initial exchange is called Authentication Service Exchange (AS) ([37], Section 3.1). It is used by the client to request an initial credential, either a ticket granting ticket (TGT) for subsequent use with the ticket granting server (TGS) (the three exchange case) or for use with a high security service (the two exchange case), from the Kerberos Authentication Server (KAS).

The last exchange is called Client/Server Authentication exchange (CS) ([37], Section 3.2), and as a prerequisite the client needs an authentication credential either directly from the authentication server (AS exchange), or from a Ticket Granting Server (TGS exchange).

The often used, but optional “middle” exchange is the Ticket Granting Service (TGS) exchange ([37], Section 3.3). Here the client needs a Ticket Granting Ticket (TGT) from an initial AS exchange to authenticate against a Ticket Granting Server, which then issues a ticket for an application server.

The three-exchange case is described in detail in the appendix. There is also a public-key variant of Kerberos which we do not consider here, since the strengths of the protocol (and its post-quantum security!) lie in the symmetric version.

1.4 Kerberos and nKerberos

In this paper, we present a formal reduction-based model for Pattern 1, which includes the well studied and important Kerberos protocol. Since reduction-based models did not consider timestamps until recently [38], we analyze an idealized version of the Kerberos protocol which replaces timestamps with nonces. We also idealize the encryption used: Our analysis assumes that all messages are encrypted using authenticated encryption, and our reduction is to this cryptographic primitive (cf [39]) and to message authentication codes.

Figure 2 shows a direct comparison between Kerberos v5 and the protocol analyzed in this paper (and subsequently denoted by nKerberos for “nonce-based Kerberos”). We tried to model nKerberos as closely as possible on Kerberos v5, to facilitate the adaption of the model and the proof presented here. The differences between both protocols are the following:

– The timestamp $t_S$ generated by the Ticket Granting Server TGS and the timestamp $t_C$ generated by the client are replaced by a nonce $n_S$ generated by the server. The generation of the nonce is triggered by an Init message to the server (which will not be included in the transcript of all messages), thus our protocol requires one additional Round Trip Time (RTT).

– The handling of identities is simplified: we always include the identities of both client and server in all messages.

– We replace the authenticated encryption of the last two messages by a MAC computation. This change is optional (the proof will still work if authenticated encryption is used) and was mainly done for didactic reasons: We wanted to stress the fact that confidentiality does not play any role here.
Fig. 2: A direct comparison between the 3-party case in Kerberos v5 and nKerberos. $AE(k; m)$ denotes authenticated encryption of message $m$ with key $k$, and $MAC(k; m)$ denotes a message authentication code.

- We split the session key $k_{CS}$ used in Kerberos v5 into two independent keys: the session key $k_{CS}$ which is the result of the 3-party delegated authenticated key exchange, and the MAC key $k_{MAC}$ which is used for key confirmation. It is important that $k_{CS}$ is not used for key confirmation, otherwise we will have to use the more complex ACCE model [40] instead of the AKE model. Instead of transmitting two keys, a standard PRF-based forking construction could be used: $k_{CS} || k_{MAC} := PRF(k; const)$.

1.5 Formal model

Our formal model only differs in the key setup from two-party AKE protocols. E.g. in [15], Bellare and Rogaway introduced the AKEP protocol, where client and server shared a (long-lived) symmetric key, and used this key to authenticate each other and to agree on a new session key. In our model, client and server do not share anything, but need the help of a trusted third party, the Kerberos Authentication Server (KAS), to establish a session key. We therefore speak of delegated AKE protocols. Our setup condition is that each party shares a long-lived symmetric key with the KAS.

The KAS itself is modelled as a public oracle: any party can call it (even the adversary), and on input two identities and two nonces it will output two authenticated ciphertexts encrypted with the long-lived keys of the two entities whose IDs were given.

We kept all other details of the formal model as close as possible on the formal model of Bellare and Rogaway [15], to enable comparison of our results with results on AKE protocols in general.
1.6 Contributions

The contributions of this paper can be summarized as follows.

– We give the first reduction-based security proof for a close variant of the Kerberos protocol.
– We propose a general security model for 3-party protocols using communication pattern 1.
– We give new insights on how pattern-1-protocols should be constructed.

2 Building blocks

We only need two building blocks in our construction of nKerberos.

2.1 Message Authentication Codes (MAC).

Informally speaking, a MAC is a cryptographic checksum that can only be generated or verified by parties who know the corresponding symmetric key; an adversary who does not know the key used in a MAC construction should not be able to generate a valid message authentication code (MAC).

This is formalized in a game between an adversary an a MAC challenger: The challenger generates a random key and uses this key to compute and verify MACs. The adversary may send polynomially many messages \( m_1, m_2, \ldots, m_q \) to the challenger, and receives valid MACs \( \text{mac}_i := \text{MAC}(k; m_i) \) for these messages. If the adversary manages to produce a MAC \( \text{mac}^* \) for a message \( m^* \in \{m_1, m_2, \ldots, m_q\} \), then he wins the game.

The MAC assumption now states that the probability \( \epsilon_{\text{MAC}} \) that any adversary wins the MAC game is negligibly small.

2.2 Authenticated encryption.

Informally speaking, an adversary should not be able to alter or forge a ciphertext (INT-CTXT), or to distinguish the contents of a ciphertext from random data (IND-CCA). Instantiations of authenticated encryption e.g. include ENCRYPT-then-MAC constructions, and the Galois Counter Mode (GCM) of block ciphers. Please refer to [41] for a detailed discussion of authenticated encryption and the relation of the different security notions.

To capture this intuition in a formal way, consider an authenticated encryption challenger that offers the two interfaces described in Figure 3 to an adversary. Since all messages issued by the KAS are of equal length, we adapted the length-hiding authenticated encryption game from [42].

The Encrypt interface models indistinguishability of plaintexts: Obviously ciphertexts must have the same length \( \text{len} \) to be indistinguishable; the corresponding plaintexts may share the same header \( H \). If the adversary can distinguish the ciphertext containing \( m_0 \) from the ciphertext containing \( m_1 \), he can predict the secret bit \( b \) of the AE challenger and win the game.
<table>
<thead>
<tr>
<th>Encrypt($m_0, m_1, \text{len}, H$):</th>
<th>Decrypt($C, H$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C^{(0)} \leftarrow \text{enc}(k; \text{len}, H, m_0)$</td>
<td>If $C$ in CList abort</td>
</tr>
<tr>
<td>$C^{(1)} \leftarrow \text{enc}(k; \text{len}, H, m_1)$</td>
<td>If $b^*_i = 0$, then return ⊥</td>
</tr>
<tr>
<td>If $C^{(0)} = ⊥$ or $C^{(1)} = ⊥$ then return $⊥$</td>
<td>$m = \text{dec}(k; H, C)$</td>
</tr>
<tr>
<td>$C := C^{(b)}$</td>
<td>Return $m$</td>
</tr>
<tr>
<td>Add $C$ to CList</td>
<td></td>
</tr>
<tr>
<td>Return $C$</td>
<td></td>
</tr>
</tbody>
</table>

Here $b$ (a bit) and $k$ (a key) are randomly chosen at the begin of the experiment, and kept fixed during the experiment.

Fig. 3: Encrypt and Decrypt oracles in the length-hiding authenticated encryption security experiment.

Less obvious is the role of the Decrypt interface: It should of course not be misused to distinguish a ciphertext on $m_0$ from a ciphertext on $m_1$, so it will not decrypt messages on CList. However, if the adversary manages to produce a valid ciphertext (e.g. a ciphertext plus a valid MAC on the ciphertext), then he can compute the secret bit of the AE challenger: if the challenger outputs $⊥$, then $b = 0$, else if the output is a valid message $m$, then $b = 1$.

In the AE security game, a challenger chooses a key $k$ and a bit $b$ randomly, and makes the Encrypt and Decrypt interfaces available to a polynomial adversary $A$. After polynomially many queries to these interfaces, the adversary eventually outputs a bit $b'$. Let $\text{Adv}_{AE}(A) := |Pr(b = b') - 1/2|$.

The authenticated encryption (AE) assumption now states that for a secure AE scheme with a randomly chosen key $k$, the advantage $\epsilon_{AE} = \max_A \{\text{Adv}_{AE}(A)\}$ in the above security game is negligibly small.

### 3 Formal Model

Formal models for authenticated key exchange typically consist of three different parts: (1) A computational model, where the implementation of the protocol is described in a precise way in the terminology of theoretical computer science, i.e. as Turing machines. Here we model e.g. global variables (managed by the party $P$), and local variables (managed by each process $\pi$). (2) An adversarial model, where we describe the attack capabilities of our adversary. Typically we exaggerate these capabilities (e.g. we assume that the adversary controls the whole network, and may compute some of the cryptographic keys used by the system). If the protocol is secure against such an exceedingly strong adversary, it will be secure against any real, weaker adversary. (3) In the security model, we must define which events we will count as “breaking” the protocol. This is a difficult task, since we have to exclude trivial attacks on the protocol which result directly from the adversarial capabilities.
3.1 Computational Model for Key Exchange

In accordance with the line of research [40, 43–46] initiated by Bellare and Rogaway [15], we model the execution environment in terms of polynomial Turing machines. Our notation closely follows [40].

**Execution environment.** Let \( P_0, \ldots, P_n \) denote the different parties involved in the cryptographic protocol. Each party \( P_i \) has several global variables, e.g. an identity \( ID_i \) (which we may omit from time to time since it is uniquely determined by the index \( i \)), and a symmetric long-lived key \( k_i \) which it may share with another party. Each party \( P_i \) may fork off processes \( \{ \pi^i_s : s \in \{1, \ldots, l\} \} \) (sometimes called process oracles or oracle in the following). These processes share the global variables of party \( P_i \), and may store new local variables in their own process memory, e.g. session keys \( k^i_s \), transcripts \( T^i_s \) and nonces \( n^i_s \).

A process \( \pi^i_s \) may be activated by the party (forking off a new process). After activation, it performs actions described in the protocol specification, and may either accept or reject in the end.

Party \( P_0 \) is the Kerberos Authentication Server (KAS). The KAS acts as a public oracle: Any party (including the adversary \( A \) may send queries to this party, and these queries will be answered. We model \( P_0 \) as a single process. W.l.o.g. we assume that there is only one KAS, and we discuss in Section 6 how to deal with more than one KAS.

Parties \( P_i, 1 \leq i \leq n \), may have either the role client or the role server during protocol execution. The adversary \( A \notin \{ P_0, \ldots, P_n \} \) is another special party that may interact with all processes by issuing different types of queries.

**Key Setup.** Before the first query is asked, long-term symmetric keys \( k_i \) for each party \( P_i, i \in \{1, \ldots, n\} \) are generated. Each key \( k_i \) is stored as a global variable at party \( P_i \), and additionally at the KAS.

**Matching conversations.** An cryptographic protocol is run between two processes \( \pi^i_s \) and \( \pi^j_t \), where each process may either “accept” or “reject” at the end of the protocol. We define correctness and security of a protocol using the concept of matching conversations as introduced by Bellare and Rogaway [15].

In the following let \( T^i_{i,s} \) denote the transcript of all messages sent and received by process \( \pi^i_s \). Intuitively, we would like to say that an authentication protocol is correct, if a process \( \pi^i_s \) outputs “accept” if there exists a process \( \pi^j_t \) with \( T^i_{i,s} = T^j_{j,t} \). Likewise, we would like to say that an authentication protocol is secure, if a process accepts only if there exists a process \( \pi^j_t \) with \( T^i_{i,s} = T^j_{j,t} \).

As pointed out in [15], we face a minor technical obstacle here, which is inherent to all protocols. Suppose that \( \pi^j_t \) sends the last message of the protocol. Then process \( \pi^i_s \) does not get any response to its last message, and thus has to accept or not without knowing whether \( \pi^j_t \) received the last message. Thus the concept of matching conversations is not symmetric, but directional. This is captured by the following definition.

\[ \text{As an alternative, session identifiers may be used, which can be defined as partial transcripts of the protocol messages.} \]
Definition 1. Let \( T_{i,s} \) denote the transcript of all messages sent and received by process \( \pi^{s}_{i} \). Let \( T_{i,s}^{(-1)} \) be the transcript \( T_{i,s} \) truncated by the last message. We say that a process \( \pi^{s}_{i} \) has a matching conversation to process \( \pi^{t}_{j} \), if

- \( \pi^{s}_{i} \) has sent the last message, and it holds that \( T_{i,s}^{(-1)} = T_{j,t}^{(-1)} \), or
- \( \pi^{t}_{j} \) has sent the last message, and it holds that \( T_{i,s} = T_{j,t} \).

We say that two processes \( \pi^{s}_{i} \) and \( \pi^{t}_{j} \) have matching conversations if \( \pi^{s}_{i} \) has a matching conversation to process \( \pi^{t}_{j} \), and vice versa.

3.2 Adversarial Model for Key Exchange

We model adversarial capabilities through different queries. The fact that our adversary controls the whole network is modelled by the Send query: Two parties/processes are never allowed to exchange messages directly, they must all pass through the adversary. The adversary then forwards these messages through Send queries to other process oracles, and receives their responses, which he again forwards via Send. If he only forwards unaltered messages to the intended destination, we call the adversary benign. A benign adversary may be used to model passive attacks on protocols. An active adversary may however alter any parameter of the message: Its destination, its content, its ordering, etc.

The fact that, in a distributed cryptographic system, the adversary may be able to learn “old” session keys (e.g. through exhaustive key search, or in the case of TLS through Bleichenbacher attacks) is modelled by the Reveal query. It will be part of the security model to define what “old” means in a mathematically correct definition.

Finally the fact that the adversary may learn “something” about a “fresh” session key is modelled by the Test query. Please notice that it would be sufficient for the adversary to learn a single bit of the session key to reliably distinguish this key from a random value. If on the other hand we can prove that he cannot distinguish the key from a random value, then the adversary will never be able to compute this key.

- Send\((P_{0}, m)\): This query can be used to send a message to the KAS \( P_{0} \), the answer will be returned to the adversary. Please note that this is the only query that can be addressed to \( P_{0} \), all other queries are restricted to indices \( 1, ..., n \).
- Send\((\pi^{s}_{i}, m)\): The active adversary can use this query to send any message \( m \) of his own choice to oracle \( \pi^{s}_{i} \). The oracle will respond according to the protocol specification. If \( m = \emptyset \), where \( \emptyset \) denotes the empty string, then \( \pi^{s}_{i} \) will respond with the first protocol message.
- Reveal\((\pi^{s}_{i})\): The adversary may learn the encryption key \( K \) computed in process \( \pi^{s}_{i} \) by asking this type of query. The adversary submits \( \pi^{s}_{i} \) to the black-box. If process \( \pi^{s}_{i} \) has “accepted”, the black-box responds with the key \( k \) in process \( \pi^{s}_{i} \). Otherwise some failure symbol \( \perp \) is returned.
– **Test**($\pi^s_i$): This query may only be asked once throughout the game. If process $\pi^s_i$ has not (yet) “accepted”, the black-box returns some failure symbol $\bot$. Otherwise the black-box flips a fair coin $b$. If $b = 0$, a random element from the keyspace is returned. If $b = 1$ then the session key $k$ computed in process $\pi^s_i$ is returned.
– **Corrupt**($P_i$): If the adversary sends this query to any of the oracles $\pi^s_i$, this oracle will answer with the secret long-lived key $k_i$ of party $P_i$. This key is then marked as corrupted. In case a party only possesses one key (i.e. it is a client or server party), then we call this party corrupted too.

### 3.3 Security Model

We define protocol security as a game where a protocol is insecure if any adversary can win the game with non-negligible advantage. This advantage may either be the probability that something “bad” happens during protocol execution, or the difference of distinguishing a key from a random value, and a random coin toss. To better describe the game, instead of using a distributed protocol environment we let a **challenger** faithfully simulate the protocol.

Consider the following security game played between a challenger $C$ and an adversary $A$.

1. The challenger generates $n + 1$ parties $P_i$, $i \in \{0, 1, ..., n\}$, where party $P_0$ is the KAS. He randomly generates $n$ symmetric keys for authenticated encryption, assigns one key $k_i$ to each party $P_i$ for $i = 1, ..., n$, and also assigns all keys to party $P_0$.
2. The adversary may ask arbitrary queries **Send**, **Reveal** and **Corrupt** to any process $\pi^s_i$, $i \in \{1, ..., n\}$, $s \in \{1, ..., \ell\}$. Queries can be made adaptively.
3. Eventually, the adversary ask a **Test** query.
4. Again, the adversary may ask the above queries.
5. Finally, the adversary outputs a bit $b'$.

The following definition only targets client or server processes, since only these only these processes can reach the state “accept”. Thus the KAS $P_0$ cannot accept maliciously by definition.

**Definition 2.** We say that a process $\pi^s_i$, $i \in \{1, ..., n\}$ accepts maliciously with intended partner $P_j$ if the messages returned during protocol execution were addressed to an uncorrupted party $P_j$, but there is no process $\pi^t_j$ with a matching conversation.

Please note that in the following definition, the two winning events only address processes of parties $P_1, ..., P_n$, i.e. client or sever processes.

**Definition 3.** Let $A$ be a PPT adversary, interacting with challenger $C$ in the security game described above. Let $b$ be the bit chosen by the challenger while answering the **Test** query, and let $b'$ be the output of $A$. We say that $A$ wins the game with probability $\epsilon$, if
Fig. 4: nKerberos: A nonce-based variant of the Kerberos protocol. Here $[m]_k$ denotes length-hiding authenticated encryption of message $m$ with key $k$, and $(m)_k$ denotes the MAC computed over message $m$ using key $k$.

- with probability $\geq \epsilon$ there is a process $\pi^*_s$ that accepts maliciously, or
- $\text{Adv}_A := |Pr(b = b') - \frac{1}{2}| \geq \epsilon$.

**Definition 4.** We say that an authenticated key exchange protocol is $\epsilon$-secure if no PPT adversary $A$ exists with winning probability greater than $\epsilon$.

### 4 A Nonce-based Kerberos Protocol

The ultimate goal of this paper is to prove that the nKerberos protocol is a secure AKE between Client and Server. To do so, we assume that the KAS is a publicly available oracle, and show that under this assumption the protocol flow between Client and Server constitutes a secure AKE in the sense of the original Definition from [15].

The KAS $T$ accepts requests from any party. These requests must be quadruples consisting of two different identities of parties registered at the KAS, and two nonces. For each such request he issues two authenticated ciphertexts. Both contain the quadruple received, plus a randomly chosen session key $k_{CS}$. This data is encrypted and MACed with the key(s) registered for the parties whose identities were contained in the request. Please note that the adversary is also allowed to make such requests.

The client $C$ receives a nonce $n_S$ from a party which he assumes (but is not sure about) to be the server application he wants to communicate with. He generates a fresh client nonce $n_C$, and forms a request quadruple from his own identity, the identity of the intended communication party, and the two nonces. Upon receiving an answer from the KAS, he decrypts and verifies the first ciphertext, and checks if it contains the request quadruple. If both checks are successful, he uses the received session key $k_{cs}$ to encrypt and MAC a message consisting of the received nonce $r_S$, and the two identities. He forwards the second ciphertext from KAS together with the freshly generated ciphertext to the intended communication partner (assumed to be server $S$).
The server \( S \) decrypts and verifies the ciphertext from the KAS, and checks if his nonce \( n_S \) is contained in the ciphertext. If both checks succeed, he stores values \( C, r_C \) and \( k_{CS} \) in local variables, and decrypts and verifies the second ciphertext with \( k_{CS} \). Then he checks if the second message also contains his own nonce \( n_S \). If this is the case, he accepts. \( S \) then encrypts and MACs the message \((S, C, n_C)\) and sends the ciphertext to \( C \).

When \( C \) receives this last ciphertext, he decrypts and verifies it, and checks if it contains the intended identities and the nonce \( n_C \) chosen by him. If these checks succeed, he accepts.

## 5 nKerberos is a secure delegated AKE

To prove the following theorem, we use the sequence of games technique to structure the proof. In this technique, the security game is slightly modified from one game to another, and a bound on the difference on the winning probabilities of the two successive games is given.

**Theorem 1.** Let \( n \) be the number of parties, let \( l \) be an upper bound on the number of processes of each party, and let \( q \) be the number of queries made to the KAS (both from legal parties and the adversary). Let \( \lambda \) and \( \nu \) be the bitlengths of the nonces and the keys, resp. Let \( \epsilon_{MAC} \) and \( \epsilon_{AE} \) be defined as in Section 2. Then nKerberos is a \( \epsilon \)-secure delegated AKE with

\[
\epsilon \leq 3 \cdot \left( 2 \cdot \frac{q^2}{2\nu} + \frac{(nl)^2}{2\lambda} + (nl) \cdot (3\epsilon_{AE} + \epsilon_{MAC}) \right).
\]

**Proof.** We have to show that client and server only accept if there is a matching conversation between them, and that the adversary cannot distinguish the real key \( k_{CS} \) from a random value. We start with the original security game (Game \( G_0 \)) played between the adversary \( A \) and a challenger that simulates nKerberos.

The proof is contained in Lemmata 1, 2 and 3, and \( \epsilon \) is an upper bound on the sum of the probabilities from the 3 Lemmata.

**Lemma 1.** The advantage of an adversary to distinguish real from random in a Test query is bounded by

\[
Adv^{Adv}_{\text{Test}} \leq 2 \cdot \frac{q^2}{2\nu} + \frac{(nl)^2}{2\lambda} + (nl) \cdot (2\epsilon_{AE}).
\]

**Proof.** We first show that the adversary cannot distinguish ciphertexts containing the real session key \( k_{CS} \) from a random value.

**Game \( G_1 \):** In this game, the challenger aborts the game if any nonce is chosen twice, or if any session key or MAC key is chosen twice. Please note that each process oracle may choose at most one nonce (thus we have at most \( nl \) nonces altogether, and that there may be at most \( \frac{nl}{2} \) sessions (and the same number of session keys needed). We may upper bound the number of session
keys with $q$, for both the MAC keys and the session keys. The probability that either of these collisions happens is upper bounded by $(nl)^2/2^{2\nu}$, thus we have

$$\text{Adv}(G_0) \leq \text{Adv}(G_1) + 2 \cdot \frac{q^2}{2^{2\nu}} + \frac{(nl)^2}{2^{2\lambda}}.$$ 

**Game $G_2$:** In this game, the challenger guesses the oracle $\pi$ which will be the target of the adversary’s Test query. This may be either a client oracle $\pi_C$, or a server oracle $\pi_S$. In the following we assume that the server targets a client oracle $\pi_C$, the other case is analogous. If the guess of the challenger was wrong, $b'$ is chosen randomly. Thus we have

$$\text{Adv}(G_1) \leq \text{Adv}(G_2).$$

**Game $G_3$:** In this game, the challenger replaces message $[C, S, n_C, n_S, k_{CS}, k_{MAC}]_{k_C}$ with $[C, S, n_C, n_S, k^*, k']_{k_C}$, for randomly chosen keys $k^*, k'$. The challenger internally keeps a key replacement table, where he adds new entries $(k^*, k_{CS})$ and $(k', k_{MAC})$. Since we have excluded key collisions in Game 1, all entries are unique.

This key replacement table is used in the following way: Whenever a process oracle successfully decrypts an authenticated ciphertext and stores key $k^*$ in its local variable $\kappa$, the challenger replaces this value with $k_{CS}$. The same happens for the value $k'$ stored in the local variable $k_{MAC}$, which will be replaced with $k_{MAC}$.

Now we argue that any adversary that is able to distinguish games $G_2$ and $G_3$ can be used to break the authenticated encryption assumption. To do so, we replace all operations involving the long-lived key $k_C$ with calls to the AE oracle. The AE adversary now sets $m_0 = (C, S, n_C, n_S, k_{CS}, k_{MAC})$ and $m_1 = (C, S, n_C, n_S, k^*, k')$. Thus if the AE challenger chooses to encrypt $m_0$, we are in Game $G_2$, and if he chooses to encrypt $m_1$, we are in Game $G_3$. Assume that our nKerberos adversary capable of distinguishing both games outputs 0 for Game $G_2$ and 1 for Game $G_3$. Then the AE adversary just outputs the same bit, and subsequently has the same advantage of breaking the AE assumption than the nKerberos adversary has to distinguish both games. Thus we have

$$\text{Adv}(G_2) \leq \text{Adv}(G_3) + \epsilon_{AE}.$$ 

**Game $G_4$:** In this game, the challenger replaces message $[C, S, n_C, n_S, k_{CS}, k_{MAC}]_{k_S}$ with $[C, S, n_C, n_S, k^*, k']_{k_S}$, for randomly chosen keys $k^*, k'$ from the previous game. The challenger uses the entries $(k^*, k_{CS})$ and $(k', k_{MAC})$ in his key replacement table as described before. With the same reduction argument as in the previous game we get

$$\text{Adv}(G_3) \leq \text{Adv}(G_4) + \epsilon_{AE}.$$ 

Now our adversary cannot distinguish real from random in a Test query: If he queries Test($\pi_C$), he will either get $k_{CS}$ (the real content of the key variable $\kappa$), or a random value. Both values have nothing to do with the ciphertexts exchanged, so the advantage of the adversary is 0.
Lemma 2. The probability $\epsilon_S$ that a server oracle accepts maliciously is bounded by

$$\epsilon_S \leq 2 \cdot \frac{q^2}{2^\nu} + \frac{(nl)^2}{2^\lambda} + (nl) \cdot (3\epsilon_{AE} + \epsilon_{MAC}).$$

Proof. Games $G_1$ to $G_4$ are identical to Lemma 1, except that in Game $G_2$ the challenger guesses which oracle will be the first to accept maliciously. If his guess was wrong, or if the first oracle to accept maliciously was a client oracle, he aborts the game.

Now in Game $G_4$, the MAC over message $(C, S, n_S)$ is computed with key $k_{MAC}$, which is randomly chosen and independent from all previous messages (which contain a different key $k'$). So any adversary who can forge this MAC can be turned into an adversary breaking the MAC assumption: Just replace all MAC computations involving key $k_{MAC}$ with calls to the MAC oracle.

Now server oracle $\pi_t^S$ will only accept if message $(C, S, n_C, n_S, k^*, k')$ can be verified with the long-lived AE key $k_S$, and if the MAC over message $(C, S, n_S)$ can be verified with the replacement key $k_{MAC}$ which was written into $\kappa_{MAC}$ instead of $k'$ by the nKerberos challenger. Since we have excluded nonce collisions in Game $G_1$, these messages must either come from an oracle that has a matching conversation, or must have been forged by the adversary.

Two cases must be distinguished:

Case 1: Only the MAC was forged, and the AE message comes from an oracle with matching conversation. In this case, any adversary succeeding in making $\pi_t^S$ accept can be turned into an adversary breaking the MAC assumption, by replacing all operation including key $k_{MAC}$ with calls to the MAC oracle as described above.

Case 2: Also the AE message was forged. In this case, any adversary succeeding in making $\pi_t^S$ accept can be turned into an adversary breaking the AE assumption, by replacing all operation including key $k_S$ with calls to the AE oracle.

Lemma 3. The probability $\epsilon_C$ that a client oracle accepts maliciously is bounded by

$$\epsilon_C \leq 2 \cdot \frac{q^2}{2^\nu} + \frac{(nl)^2}{2^\lambda} + (nl) \cdot (3\epsilon_{AE} + \epsilon_{MAC}).$$

Proof. Games $G_1$ to $G_4$ are identical to Lemma 1, except that in Game $G_2$ the challenger guesses which oracle will be the first to accept maliciously. If his guess was wrong, or if the first oracle to accept maliciously was a server oracle, he aborts the game.

Please note that now we know that no server oracle accepted maliciously before our client oracle.

Thus with a similar argument as in Lemma 2 we can argue that either only the last MAC message must be forged, or also the AE message from the KAS to the client must be forged. In both cases the same reductions as above apply.
6 Extension of the results

6.1 Modelling more than one Kerberos server

If each party $P_i$ shares a key with exactly one KAS, then introducing several KAS simply results in a partition of the set of all parties. This would allow us to define a Corrupt query for KAS, but after a simple change in the security definition (which excludes protocol executions with the help of corrupted KAS from the list of winning events) the security proof need not be changed.

So the only interesting case is that at least one party is assigned two keys, which it shares with different KAS. In this case, in addition to guessing the test oracle or the oracle that accepts maliciously, we would also have to guess which KAS was involved, and thus we would loose a constant factor, namely the number of different KAS, in our reduction. Since this does not change the success probability of any adversary significantly, nKerberos still remains secure.

6.2 Extending the proof to the three-exchange case

The 4-party, 3-exchange version of nKerberos is depicted in Figure 5. It needs 2 additional messages to provide the nonces from the TGS and the server, and one additional exchange between client and TGS. This additional overhead clearly shows the advantages of traditional Kerberos using timestamps over the nonce-based version as far as performance is concerned (cf. Figure 6).

The 4-party case consist of two similar 3-party protocols (client–KAS–TGS and client–TGS–server). Our security proof directly applies to the first of these 3-party protocols, if we model the (different) TGS as servers. This first protocol also establishes a symmetric key between client and TGS.

In the second 3-party protocol, the proof can easily be adapted if we adjust the key setup conditions: Here the servers share distinct long-lived symmetric keys with the TGS that is serving them.

7 Related Work

A search for “kerberos” in the computer science bibliography database (http://dblp.uni-trier.de/search?q=kerberos) returns 115 results. Most of these papers describe modifications of the Kerberos protocol, either enhancements of certain subprotocols (e.g. replacing passwords with smartcards), or combinations with other protocols.

The very first (informal) security analysis of Kerberos was given by Steven M. Bellovin and Michael Merritt [47].

There are several papers containing a security analysis of Kerberos or its variants, and most of them contain a Dolev-Yao style analysis. Kirsal et al. [33] describe an analysis of a Kerberos variant using Communication Sequential Processes (CSP) and Rank functions. CSP is also used in the analysis of Li et al. [10]. Yongjian and Pang [12] extend the Strand spaces model with timestamps,
nKerberos – 4 Party Setting

<table>
<thead>
<tr>
<th>Client</th>
<th>AS</th>
<th>TGS</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>nTGS</td>
<td>C,TGS,n'TGS</td>
<td>AE(kCTGS; C,TGS,n'TGS,kMAC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AE(kTGS; C,TGS,n'TGS,kMAC)</td>
<td>AE(kCTGS; C,TGS,n'TGS,kMAC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC(kMAC; C,TGS,nTGS)</td>
<td>MAC(kMAC; C,TGS,nTGS)</td>
</tr>
<tr>
<td>Init</td>
<td>nS</td>
<td>C,S,nC,nS</td>
<td>AE(kCS; C,S,nC,kMAC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AE(kCS; C,S,nC,kMAC)</td>
<td>AE(kCS; C,S,nC,kMAC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAC(kMAC; C,S,nC)</td>
<td>MAC(kMAC; C,S,nC)</td>
</tr>
</tbody>
</table>

Fig. 5: nKerberos extended to the 4-party case.
and apply this extended model to Kerberos. A previous analysis was given in [7].
Abdelmajid et al. [11] use a BAN Logic variant in their analysis, thereby using a
similar approach as Fan et al. [9]. Using formal analysis tools [8], Cerversato et
al. [48] found a man-in-the-middle attack in PKINIT, the public key extension
of Kerberos. The MSR language is chosen in the formal analysis performed by
Butler et al. [6]. Giampaolo Bella analyzed, with various co-authors, Version 4
in a series of papers [3–5].

A security analysis in the abstract Dolev-Yao style which also considers cryp-
tographic soundness is presented by Backes et al. [24]. In this model, crypto-
graphic primitives are treated as abstract objects, but their implementation is
shown to indeed fulfill the assumptions of the formal model. Please note that
although cryptographic soundness is a goal of this paper, it does not contain a
reduction-based proof as proposed in [15] and [44].

Yu et al. show that Kerberos v4 uses weak encryption, which enables an
attacker to perform a chosen-plaintext attack to impersonate arbitrary prin-
cipals [49]. Boldyreva and Kumar [39] confirm that the “general profile” used in
Kerberos v5 does not provide authenticated encryption, but they also propose
a slightly modified profile that is IND-CCA and IND-CTXT secure. Together
both papers indicate that the reduction-based proof given in this paper only
could be applied to the modified general profile given in [39].

8 Future work

The present paper has shown the feasability of reduction-based proofs for three-
party protocols using the Kerberos communication pattern. One basic idea of
the proof given in this paper was to model the KAS as a public oracle.

To give a complete proof of the Kerberos protocol, this idea can be adapted.
It remains to show that timestamps can be integrated into a reduction-based
security proof, e.g. along the lines of [38].

For SSO protocols, a different type of proof must be developed: Here it is
essential that the browser authenticates itself to the identity provider, so the
IdP cannot be modelled as a public oracle.

References

1. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay, “Cryptograph-
ically sound security proofs for basic and public-key kerberos,” in ESORICS 2006: 11th
European Symposium on Research in Computer Security, ser. Lecture Notes in
383.
mechanized proofs for basic and public-key Kerberos,” in ASIACCS 08: 3rd ACM
Symposium on Information, Computer and Communications Security, M. Abe and


A Cryptographic description of Kerberos

In this appendix we describe the complete Kerberos protocol in basic authentication mode. Recall that Kerberos, strictly speaking, may be a three-party or a four-party protocol, between a client, a Kerberos Authentication Server (KAS), a Ticket Granting Server (TGS), and a server. The KAS and TGS together play the role of the identity provider. The message flow of the Kerberos protocol appears in Figure 6.

The splitting of the key management functionality to two parties (KAS and TGS) has several practical advantages:

– The client has to use his long-lived secret to decrypt the message received from the KAS. In many cases, this secret is derived from a password on a computer that is not fully trusted. By splitting the functionality, this long-lived secret can immediately be erased from memory after decrypting the KAS message. All subsequent messages are encrypted with $k_{C,TGS}$, which has only a limited lifetime, and where a leakage wouldn’t affect system security too much.

– The KAS only has to answer one message per validity period from each client, thus security can have priority over performance. The TGS on the other hand only handles short-lived secrets, so here performance may get priority.

In the full Kerberos protocol, the protocol flow is as follows:

– The client $C$ sends his identity, the identity of the TGS and a nonce $n_1$ as a first message. In response, he receives two cryptograms containing the session key with the TGS: The ticket granting ticket (TGT), which is destined for the TGT, and the first cryptogram $ct_1$ which contains the nonce $n_1$ (to guarantee freshness of the message), the session key $k_{C,TGS}$, a timestamp issued by KAS, and the (repeated) identity of TGS.

– Now the client forwards the TGT, a cryptogram under the session key which contains the client’s identity and a client timestamp, the identity of the target server and a new nonce $n_3$. The response of the TGT is similar to the response of the KAS: It contains a server ticket $TS$ destined for the server, and a cryptogram $ct_3$ containing a new session key, the nonce $n_3$, a timestamp, and the identity of the server.

– In the last exchange the client forwards $ST$ and a new cryptogram $ct_4$ to the target server, who checks both values and in case of acceptance returns an encryption of the client timestamp.

This four-party message flow is an adaption of Kerberos to practical requirements. Both the ancestor of Kerberos (the Needham-Schroeder protocol) and the descendants of Kerberos (Microsoft Passport, OpenID, SAML Web Authentication) only use a three-party setting. We therefore proved security only for this 3-party setting, because our proof should be adaptable to all these other 3-party cases.
Fig. 6: Kerberos protocol with basic authentication
An extension of the proof to the four-party case was sketched (for nKerberos) in Section 6. In this full proof, the role of the timestamps must be considered: They serve as a replacement for nonces, but have different security properties. E.g., the TGT and ST can be used several times to authenticate the client, so the notion of “maliciously accepts” must be adapted, too.