

Benutzer an der Eingabeaufforderung eingegebenen Passworts zu entschlüsseln. Ist das Passwort korrekt, erhält das Anmeldeprogramm den Sitzungsschlüssel und den Einmalstempel. Es überprüft den Einmalstempel und speichert den Sitzungsschlüssel mit dem Ticket für die weitere Kommunikation mit dem TGS. Jetzt kann das Anmeldeprogramm das Passwort des Benutzers aus seinem Speicher löschen, weil für die Authentifizierung des Benutzers weiterhin das Ticket verwendet wird. Eine Anmeldesitzung wird für den Benutzer auf der Workstation gestartet. Beachten Sie, dass das Passwort des Benutzers niemals Lauschangriffen im Netzwerk ausgesetzt ist – es bleibt auf der Workstation und wird nach der Eingabe relativ schnell wieder gelöscht.

**Zugriff auf Server mit Kerberos** Immer wenn ein Programm auf einer Workstation ausgeführt wird, um auf einen neuen Dienst zuzugreifen, fordert es vom Ticket erteilenden Server ein Ticket für den Dienst an. Will sich beispielsweise ein UNIX-Benutzer an einem entfernten Computer anmelden, erhält das *rlogin*-Befehlsprogramm auf der Workstation des Benutzers ein Ticket vom Ticket erteilenden Dienst von Kerberos, um auf den *rlogind*-Netzwerkdienst zuzugreifen. Das *rlogin*-Befehlsprogramm sendet das Ticket zusammen mit einer neuen Authentifizierung in einer Anforderung an den *rlogind*-Prozess auf dem Computer, an dem sich der Benutzer anmelden will. Das *rlogind*-Programm entschlüsselt das Ticket mit dem geheimen Schlüssel des *rlogin*-Dienstes und überprüft die Gültigkeit des Tickets (d.h. ob seine Lebenszeit noch nicht abgelaufen ist). Server-Maschinen müssen darauf achten, die geheimen Schlüssel an Positionen abzulegen, auf die Eindringlinge keinen Zugriff haben.

Das *rlogind*-Programm verwendet den im Ticket enthaltenen Sitzungsschlüssel zur Entschlüsselung der Authentifizierung und überprüft, ob diese unbenutzt ist (Authentifizierungen können nur ein einziges Mal verwendet werden). Nachdem das *rlogind*-Programm sich überzeugt hat, dass das Ticket und die Authentifizierung gültig sind, müssen Name und Passwort des Benutzers nicht mehr überprüft werden, weil die ID des Benutzers dem *rlogind*-Programm bekannt ist und für diesen Benutzer auf der entfernten Maschine eine Anmeldesitzung eingerichtet wurde.

**Implementierung von Kerberos** Kerberos wird als Server implementiert, der auf einer sicheren Maschine ausgeführt wird. Für die Verwendung durch Client-Applikationen und Dienste werden mehrere Bibliotheken bereitgestellt. Der DES-Verschlüsselungs-Algorithmus wird verwendet, aber als separates Modul implementiert, das ganz einfach ersetzt werden kann.

Der Kerberos-Dienst ist skalierbar – die Welt ist in mehrere Domänen einzelner Authentifizierungsautoritäten unterteilt, so genannte *Realms*, die jeweils einen eigenen Kerberos-Server betreiben. Die meisten Prinzipale sind nur in einem Realm registriert, aber Ticket erteilende Kerberos-Server sind in allen Realms registriert. Prinzipale können sich über ihren lokalen Ticket erteilenden Server selbst den Servern in anderen Realms authentifizieren.

Innerhalb eines Realms kann es mehrere Authentifizierungs-Server geben, die alle Kopien derselben Authentifizierungs-Datenbank besitzen. Die Authentifizierungs-Datenbank wird durch eine einfache Master-/Slave-Technik repliziert. Aktualisierungen werden von einem einzelnen KDBM-Dienst (Kerberos Database Management) auf die Master-Kopie angewendet, die nur auf der Master-Maschine ausgeführt wird. Der DKBM verarbeitet Anforderungen von Benutzern, ihre Passwörter zu ändern, und Anforderungen von Systemadministratoren, Prinzipale hinzuzufügen oder zu löschen und ihre Passwörter zu ändern.

Um den Benutzern dieses Schema transparent zu machen, sollte die Lebensdauer von TGS-Tickets solange wie die längstmögliche Anmeldesitzung sein, weil die Ver-

wendung eines abgelaufenen Tickets zum Zurückweisen der Dienstanforderungen führt und die einzige Abhilfe für den Benutzer ist, die Anmeldesitzung erneut zu authentifizieren und dann neue Server-Tickets für alle verwendeten Dienste anzufordern. In der Praxis wird eine Ticket-Lebensdauer von etwa zwölf Stunden verwendet.

**Kritik an Kerberos** Das oben beschriebene Protokoll für Kerberos Version 5 enthält mehrere Verbesserungen, die Kritikpunkte an früheren Versionen aufgenommen haben Bellovin und Merritt [1990, Burrows *et al.* 1990]. Die wichtigsten Kritikpunkte an Version 4 waren, dass die in Authentifizierungen verwendeten Einmalstempel als Zeitstempel implementiert waren und dass der Schutz gegen die Wiedervorlage einer Authentifizierung damit von einer zumindest lockeren Synchronisierung der Uhren der Clients und Server abhängig war. Darüber hinaus musste das Synchronisierungs-Protokoll selbst sicher gegenüber Sicherheitsangriffen sein, wenn es verwendet werden sollte. Weitere Informationen über Protokolle zur Uhrsynchroneisierung finden Sie in Kapitel 10.

Die Protokolldefinition für Version 5 erlaubt, dass die Einmalstempel in Authentifizierungen als Zeitstempel oder als Folgenummern implementiert werden. In beiden Fällen ist es erforderlich, dass sie eindeutig sind und dass die Server eine Liste zuvor von den einzelnen Clients erhaltenen Zeitstempel führen, um sicherzustellen, dass sie nicht wiederholt vorgelegt werden. Das ist eine unbequeme Forderung an die Implementierung und es ist schwer zu realisieren, dass die Server auch im Fehlerfall Sicherheit garantieren. Kehne *et al.* [1992] haben eine Verbesserung am Kerberos-Protokoll vorgeschlagen, die nicht auf synchronisierten Uhren basiert.

Die Sicherheit von Kerberos ist von einer begrenzten Lebensdauer der Sitzungen abhängig – die Gültigkeitsdauer von TGS-Tickets ist normalerweise auf wenige Stunden begrenzt; das Zeitintervall sollte also groß genug gewählt werden, um unpraktische Unterbrechungen der Dienste zu vermeiden, aber kurz genug, um sicherzustellen, dass Benutzer, deren Registrierung gelöscht wurde oder deren Berechtigungen verringert wurden, die Ressourcen nur noch eine kurze Zeit lang nutzen können. Das könnte in einigen kommerziellen Umgebungen problematisch sein, weil die nachfolgende Anforderung an den Benutzer, zu einem beliebigen Zeitpunkt der Interaktion neue Authentifizierungsdetails bereitzustellen, Probleme für die Applikation darstellen könnte.

### 7.6.3 Sicherung elektronischer Transaktionen durch sichere Sockets

Das SSL-Protokoll (Secure Socket Layer) wurde ursprünglich von der Netscape Corporation [Netscape 1996] entwickelt und als Standard vorgeschlagen, der insbesondere die oben beschriebenen Bedürfnisse abdecken sollte. Eine erweiterte Version von SSL wurde als Internetstandard unter dem Namen TLS-Protokoll (Transport Layer Security) übernommen, beschrieben in RFC 2246 [Dierk und Allen 1999]. SSL wird von den meisten Browsern unterstützt und im E-Commerce ganz allgemein eingesetzt. Seine wichtigsten Funktionsmerkmale sind:

**Abstimmbare Verschlüsselungs- und Authentifizierungs-Algorithmen** In einem offenen Netzwerk sollten wir nicht davon ausgehen, dass alle Teilnehmer dieselbe Client-Software verwenden oder dass die gesamte Client- und Server-Software einen bestimmten Verschlüsselungs-Algorithmus enthält. Die Gesetze einiger Länder versuchen, die Verwendung bestimmter Verschlüsselungs-Algorithmen nur auf diese Länder zu beschränken. SSL wurde so entwickelt, dass die für die Verschlüsselung

und Authentifizierung verwendeten Algorithmen während des anfänglichen Handshakes zwischen den Parteien an den beiden Enden der Verbindung abgestimmt werden. Es kann sich herausstellen, dass sie nicht ausreichend viele Algorithmen gemeinsam haben, dann schlägt der Verbindungsversuch fehl.

**Sichere Kommunikation nach Bootstrapping** Um den Bedarf nach einer sicheren Kommunikation ohne vorherige Abstimmung oder Hilfe von Dritten abzudecken, wird der sichere Kanal mithilfe eines Protokolls eingerichtet, das ähnlich dem zuvor beschriebenen hybriden Schema ist. Die nicht verschlüsselte Kommunikation wird für den anfänglichen Austausch verwendet, anschließend die Verschlüsselung mit öffentlichem Schlüssel und schließlich mit geheimem Schlüssel, nachdem ein gemeinsamer geheimer Schlüssel eingerichtet wurde. Jeder Wechsel ist optional und erfolgt nur nach vorhergehender Einigung.

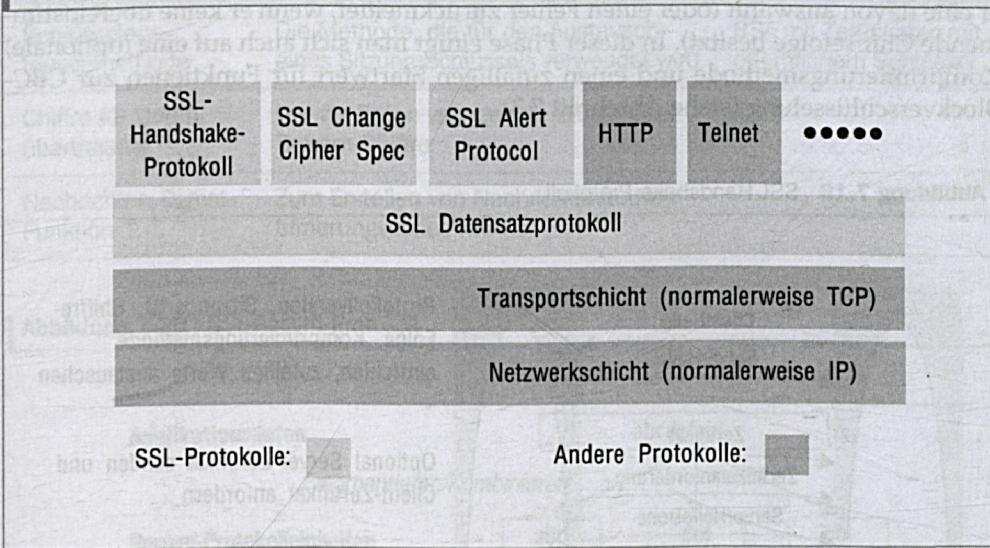
Der sichere Kanal ist also vollständig konfigurierbar, sodass Kommunikation in jede Richtung verschlüsselt und authentifiziert werden kann aber nicht muss und die Rechen-Ressourcen nicht für unnötige Verschlüsselung verbraucht werden müssen.

Die Details des SSL-Protokolls werden veröffentlicht und standardisiert und es gibt mehrere Software-Bibliotheken und Toolkits, die sie unterstützen [Hirsch 1997, [www.opensl.org](http://www.opensl.org)]. Einige davon sind Public Domain. Es hat sich in einem breiten Bereich an Applikations-Software eingebürgert und seine Sicherheit wurde unabhängig überprüft.

SSL besteht aus zwei Schichten (Abbildung 7.18), einer SSL Record Protocol-Schicht, die einen sicheren Kanal implementiert und Nachrichten verschlüsselt und authentifiziert, die über ein beliebiges verbindungsorientiertes Protokoll übertragen werden, und einer Handshake-Schicht, die das SSL Handshake-Protokoll und zwei weitere verwandte Protokolle, die eine SSL-Sitzung (d.h. einen sicheren Kanal) zwischen einem Client und einem Server einrichten, enthält. Beide werden normalerweise durch Software-Bibliotheken auf Applikationsebene im Client und im Server implementiert. Das SSL Record-Protokoll ist eine Schicht auf Sitzungsebene; es kann genutzt werden, um Daten auf Applikationsebene zwischen einem Prozesspaar transparent zu transportieren, deren Geheimhaltung, Integrität und Authentizität garantiert wird. Dies sind genau die Eigenschaften, die wir in unserem Sicherheitsmodell (Abschnitt 2.3.3) für sichere Kanäle spezifiziert haben, aber in SSL können die kommunizierenden Partner auswählen, ob sie die Verschlüsselung und Authentifizierung von Nachrichten in jeder Richtung anwenden wollen. Jede sichere Sitzung erhält einen ID und jeder Partner kann Sitzungs-IDs für die nachfolgende Wiederverwendung in einem Cache ablegen; er vermeidet damit den Aufwand, eine neue Sitzung einzurichten zu müssen, wenn eine weitere sichere Sitzung mit demselben Partner benötigt wird.

SSL wird allgemein eingesetzt, um eine sichere Kommunikationsschicht zwischen existierenden Protokollen auf Applikationsebene einzufügen. Das Protokoll wurde zuerst 1994 veröffentlicht und nach Überarbeitungen entstanden zwei Nachfolgerversionen. SSL 3.0 ist Thema eines Standardisierungsvorschlags [Netscape 1996]. Es wird wohl hauptsächlich eingesetzt, um HTTP-Interaktionen für den Internet-Kommerz und andere sicherheitssensible Applikationen zu sichern. Es wurde von fast allen Webbrowsern und Webservern implementiert: die Verwendung des Protokoll-Präfixes *https:* in URLs initiiert die Einrichtung eines sicheren SSL-Kanals zwischen einem Browser und einem Webserver. Außerdem wird es häufig eingesetzt, um sichere Implementierungen von Telnet, FTP und vielen anderen Applikations-Protokollen zu realisieren. SSL ist der *De-facto*-Standard für Applikationen, die sichere

**Abbildung 7.17** SSL-Protokollstapel (die Abbildungen in diesem Abschnitt basieren auf Diagrammen aus [Hirsch 1997] und werden mit freundlicher Genehmigung von Frederick Hirsch veröffentlicht)



Kanäle benötigen, und es gibt eine große Auswahl sowohl kommerzieller als auch Public-Domain-Implementierungen mit APIs für CORBA und Java.

Abbildung 7.19 zeigt das SSL-Handshake-Protokoll. Der Handshake erfolgt über eine bereits existierende Verbindung. Er beginnt im Client und richtet eine SSL-Sitzung ein, indem er die abgestimmten Optionen und Parameter für die Verschlüsselung und Authentifizierung austauscht. Die Handshake-Abfolge variiert abhängig davon, welche Authentifizierung für Client und Server benötigt wird. Das Handshake-Protokoll kann auch später aufgerufen werden, um die Spezifikation eines sicheren Kanals zu ändern; beispielsweise kann die Kommunikation mit einer Nachrichtenauthentifizierung nur unter Verwendung von Nachrichtenauthentifizierungscodes (MACs) erfolgen. Später kann eine Verschlüsselung hinzugefügt werden. Das erreicht man, indem man das Handshake-Protokoll erneut ausführt, um unter Verwendung des existierenden Kanals eine neue Chiffrespezifizierung abzustimmen.

Der erste SSL-Handshake ist empfindlich gegenüber Mann-in-der-Mitte-Angriffen, wie in Abschnitt 7.2.2 im Szenario 3 beschrieben. Um sich dagegen zu schützen, kann der öffentliche Schlüssel, mit dem das erste empfangene Zertifikat überprüft wird, über einen separaten Kanal ausgeliefert werden – beispielsweise können Browser und andere Internet-Software, die auf CD-ROM ausgeliefert werden, mehrere öffentliche Schlüssel für einige bekanntere Zertifikatautoritäten enthalten. Eine weitere Maßnahme für die Clients bekannter Dienste ist, die Domänennamen der Dienste in die Zertifikate mit öffentlichem Schlüssel aufzunehmen – die Clients sollten nur mit dem Dienst unter der IP-Adresse arbeiten, die diesem Domänennamen entspricht.

SSL unterstützt eine Vielzahl von Optionen für die zu verwendenden Verschlüsslungsfunktionen. Man spricht häufig auch von einer *Chiffrefolge*. Eine Chiffrefolge beinhaltet je eine Auswahl für jedes der in Abbildung 7.19 gezeigten Funktionsmerkmale.

Benutzer an der Eingabeaufforderung eingegebenen Passworts zu entschlüsseln. Ist das Passwort korrekt, erhält das Anmeldeprogramm den Sitzungsschlüssel und den Einmalstempel. Es überprüft den Einmalstempel und speichert den Sitzungsschlüssel mit dem Ticket für die weitere Kommunikation mit dem TGS. Jetzt kann das Anmeldeprogramm das Passwort des Benutzers aus seinem Speicher löschen, weil für die Authentifizierung des Benutzers weiterhin das Ticket verwendet wird. Eine Anmeldesitzung wird für den Benutzer auf der Workstation gestartet. Beachten Sie, dass das Passwort des Benutzers niemals Lauschangriffen im Netzwerk ausgesetzt ist – es bleibt auf der Workstation und wird nach der Eingabe relativ schnell wieder gelöscht.

**Zugriff auf Server mit Kerberos** Immer wenn ein Programm auf einer Workstation ausgeführt wird, um auf einen neuen Dienst zuzugreifen, fordert es vom Ticket erteilenden Server ein Ticket für den Dienst an. Will sich beispielsweise ein UNIX-Benutzer an einem entfernten Computer anmelden, erhält das *rlogin*-Befehlsprogramm auf der Workstation des Benutzers ein Ticket vom Ticket erteilenden Dienst von Kerberos, um auf den *rlogind*-Netzwerkdienst zuzugreifen. Das *rlogin*-Befehlsprogramm sendet das Ticket zusammen mit einer neuen Authentifizierung in einer Anforderung an den *rlogind*-Prozess auf dem Computer, an dem sich der Benutzer anmelden will. Das *rlogind*-Programm entschlüsselt das Ticket mit dem geheimen Schlüssel des *rlogin*-Dienstes und überprüft die Gültigkeit des Tickets (d.h. ob seine Lebenszeit noch nicht abgelaufen ist). Server-Maschinen müssen darauf achten, die geheimen Schlüssel an Positionen abzulegen, auf die Eindringlinge keinen Zugriff haben.

Das *rlogind*-Programm verwendet den im Ticket enthaltenen Sitzungsschlüssel zur Entschlüsselung der Authentifizierung und überprüft, ob diese unbenutzt ist (Authentifizierungen können nur ein einziges Mal verwendet werden). Nachdem das *rlogind*-Programm sich überzeugt hat, dass das Ticket und die Authentifizierung gültig sind, müssen Name und Passwort des Benutzers nicht mehr überprüft werden, weil die ID des Benutzers dem *rlogind*-Programm bekannt ist und für diesen Benutzer auf der entfernten Maschine eine Anmeldesitzung eingerichtet wurde.

**Implementierung von Kerberos** Kerberos wird als Server implementiert, der auf einer sicheren Maschine ausgeführt wird. Für die Verwendung durch Client-Applikationen und Dienste werden mehrere Bibliotheken bereitgestellt. Der DES-Verschlüsselungs-Algorithmus wird verwendet, aber als separates Modul implementiert, das ganz einfach ersetzt werden kann.

Der Kerberos-Dienst ist skalierbar – die Welt ist in mehrere Domänen einzelner Authentifizierungsautoritäten unterteilt, so genannte *Realms*, die jeweils einen eigenen Kerberos-Server betreiben. Die meisten Prinzipale sind nur in einem Realm registriert, aber Ticket erteilende Kerberos-Server sind in allen Realms registriert. Prinzipale können sich über ihren lokalen Ticket erteilenden Server selbst den Servern in anderen Realms authentifizieren.

Innerhalb eines Realms kann es mehrere Authentifizierungs-Server geben, die alle Kopien derselben Authentifizierungs-Datenbank besitzen. Die Authentifizierungs-Datenbank wird durch eine einfache Master-/Slave-Technik repliziert. Aktualisierungen werden von einem einzelnen KDBM-Dienst (Kerberos Database Management) auf die Master-Kopie angewendet, die nur auf der Master-Maschine ausgeführt wird. Der KDBM verarbeitet Anforderungen von Benutzern, ihre Passwörter zu ändern, und Anforderungen von Systemadministratoren, Prinzipale hinzuzufügen oder zu löschen und ihre Passwörter zu ändern.

Um den Benutzern dieses Schema transparent zu machen, sollte die Lebensdauer von TGS-Tickets solange wie die längstmögliche Anmeldesitzung sein, weil die Ver-

wendung eines abgelaufenen Tickets zum Zurückweisen der Dienstanforderungen führt und die einzige Abhilfe für den Benutzer ist, die Anmeldesitzung erneut zu authentifizieren und dann neue Server-Tickets für alle verwendeten Dienste anzufordern. In der Praxis wird eine Ticket-Lebensdauer von etwa zwölf Stunden verwendet.

**Kritik an Kerberos** Das oben beschriebene Protokoll für Kerberos Version 5 enthält mehrere Verbesserungen, die Kritikpunkte an früheren Versionen aufgenommen haben Bellovin und Merritt [1990, Burrows *et al.* 1990]. Die wichtigsten Kritikpunkte an Version 4 waren, dass die in Authentifizierungen verwendeten Einmalstempel als Zeitstempel implementiert waren und dass der Schutz gegen die Wiedervorlage einer Authentifizierung damit von einer zumindest lockeren Synchronisierung der Uhren der Clients und Server abhängig war. Darüber hinaus musste das Synchronisierungs-Protokoll selbst sicher gegenüber Sicherheitsangriffen sein, wenn es verwendet werden sollte. Weitere Informationen über Protokolle zur Uhrsynchrosnchronisierung finden Sie in Kapitel 10.

Die Protokolldefinition für Version 5 erlaubt, dass die Einmalstempel in Authentifizierungen als Zeitstempel oder als Folgenummern implementiert werden. In beiden Fällen ist es erforderlich, dass sie eindeutig sind und dass die Server eine Liste zuvor von den einzelnen Clients erhaltenen Zeitstempel führen, um sicherzustellen, dass sie nicht wiederholt vorgelegt werden. Das ist eine unbequeme Forderung an die Implementierung und es ist schwer zu realisieren, dass die Server auch im Fehlerfall Sicherheit garantieren. Kehne *et al.* [1992] haben eine Verbesserung am Kerberos-Protokoll vorgeschlagen, die nicht auf synchronisierten Uhren basiert.

Die Sicherheit von Kerberos ist von einer begrenzten Lebensdauer der Sitzungen abhängig – die Gültigkeitsdauer von TGS-Tickets ist normalerweise auf wenige Stunden begrenzt; das Zeitintervall sollte also groß genug gewählt werden, um unpraktische Unterbrechungen der Dienste zu vermeiden, aber kurz genug, um sicherzustellen, dass Benutzer, deren Registrierung gelöscht wurde oder deren Berechtigungen verringert wurden, die Ressourcen nur noch eine kurze Zeit lang nutzen können. Das könnte in einigen kommerziellen Umgebungen problematisch sein, weil die nachfolgende Anforderung an den Benutzer, zu einem beliebigen Zeitpunkt der Interaktion neue Authentifizierungsdetails bereitzustellen, Probleme für die Applikation darstellen könnte.

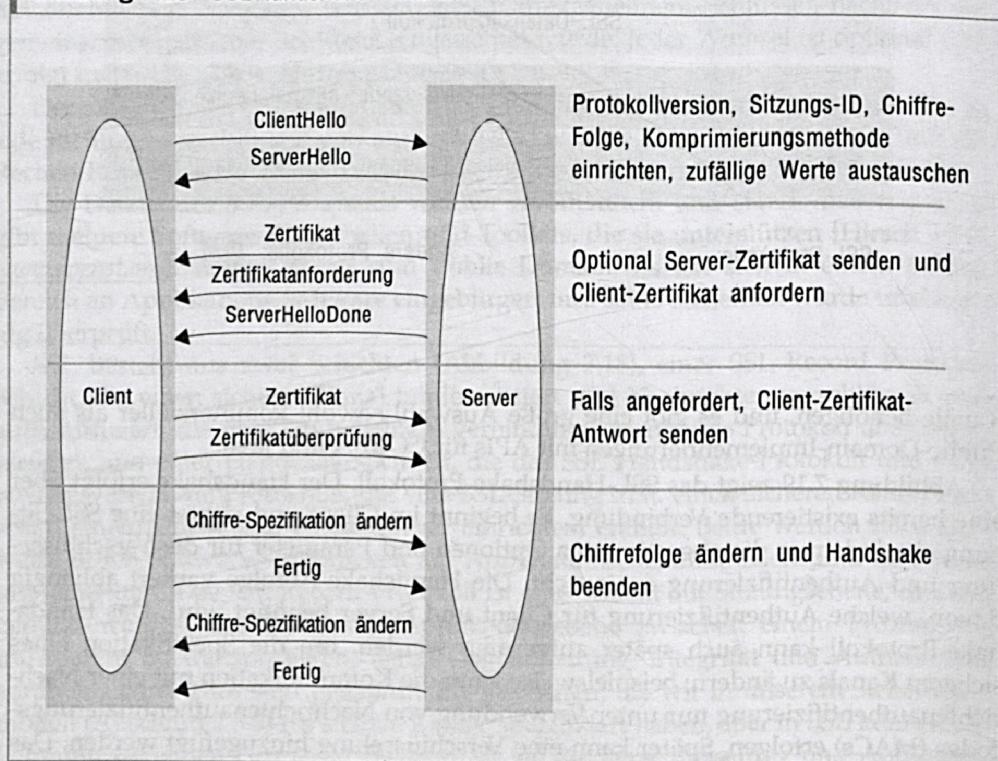
### 7.6.3 Sicherung elektronischer Transaktionen durch sichere Sockets

Das SSL-Protokoll (Secure Socket Layer) wurde ursprünglich von der Netscape Corporation [Netscape 1996] entwickelt und als Standard vorgeschlagen, der insbesondere die oben beschriebenen Bedürfnisse abdecken sollte. Eine erweiterte Version von SSL wurde als Internetstandard unter dem Namen TLS-Protokoll (Transport Layer Security) übernommen, beschrieben in RFC 2246 [Dierk und Allen 1999]. SSL wird von den meisten Browsern unterstützt und im E-Commerce ganz allgemein eingesetzt. Seine wichtigsten Funktionsmerkmale sind:

**Abstimmbare Verschlüsselungs- und Authentifizierungs-Algorithmen** In einem offenen Netzwerk sollten wir nicht davon ausgehen, dass alle Teilnehmer dieselbe Client-Software verwenden oder dass die gesamte Client- und Server-Software einen bestimmten Verschlüsselungs-Algorithmus enthält. Die Gesetze einiger Länder versuchen, die Verwendung bestimmter Verschlüsselungs-Algorithmen nur auf diese Länder zu beschränken. SSL wurde so entwickelt, dass die für die Verschlüsselung

In Client und Server sind verschiedene bekannte Chiffrefolgen mit Standard-IDs vorab geladen. Während des Handshakes zeigt der Server dem Client eine Liste der Chiffrefolgen-IDs an, die ihm zur Verfügung stehen, und der Client antwortet, indem er eine davon auswählt (oder einen Fehler zurückmeldet, wenn er keine übereinstimmende Chiffrefolge besitzt). In dieser Phase einigt man sich auch auf eine (optionale) Komprimierungsmethode und einen zufälligen Startwert für Funktionen zur CBC-Blockverschlüsselung (siehe Abschnitt 7.3).

**Abbildung 7.18** SSL-Handshake-Protokoll



Anschließend authentifizieren sich die Partner optional gegenseitig, indem sie Zertifikate mit öffentlichem Schlüssel im X.509-Format austauschen. Diese Zertifikate können von einer Autorität für öffentliche Schlüssel erhalten oder einfach temporär für genau diesen Zweck erzeugt werden. In jedem Fall muss es mindestens einen öffentlichen Schlüssel für die nächste Phase des Handshakes geben.

Ein Partner erzeugt dann ein *Pre-Master-Geheimnis* und sendet es mit dem öffentlichen Schlüssel verschlüsselt an den anderen Partner. Ein Pre-Master-Geheimnis ist ein großer Zufallswert, der von beiden Partnern verwendet wird, um die beiden Sitzungsschlüssel (so genannte Schreib-Schlüssel) für die Verschlüsselung von Daten in beide Richtungen und die Geheimnisse für die Nachrichtenauthentifizierung zu erzeugen. Nachdem all das erfolgt ist, beginnt eine sichere Sitzung. Sie wird ausgelöst durch die zwischen den Partnern ausgetauschten *ChangeCipherSpec*-Nachrichten. Ihnen folgen *Finished*-Nachrichten. Nachdem die *Finished*-Nachrichten ausgetauscht wurden, wird jede weitere Kommunikation verschlüsselt und gemäß der ausgewählten Chiffrefolge mit den abgestimmten Schlüsseln verschlüsselt.

**Abbildung 7.19** Konfigurationsoptionen für den SSL-Handshake

| Komponente                   | Beschreibung   | Beispiel  |
|------------------------------|--|---|
| Schlüsselaustauschmethode    | Die Methode, die für den Austausch eines Sitzungsschlüssels verwendet wird | RSA mit Zertifikaten mit öffentlichem Schlüssel |
| Chiffre für Datenübertragung | Der für Daten verwendete Block- oder Stream-Chiffre                        | IDEA  |
| Nachrichten-Digest-Funktion  | Zum Erstellen von Nachrichtenauthentifizierungscodes (MACs)                |   |

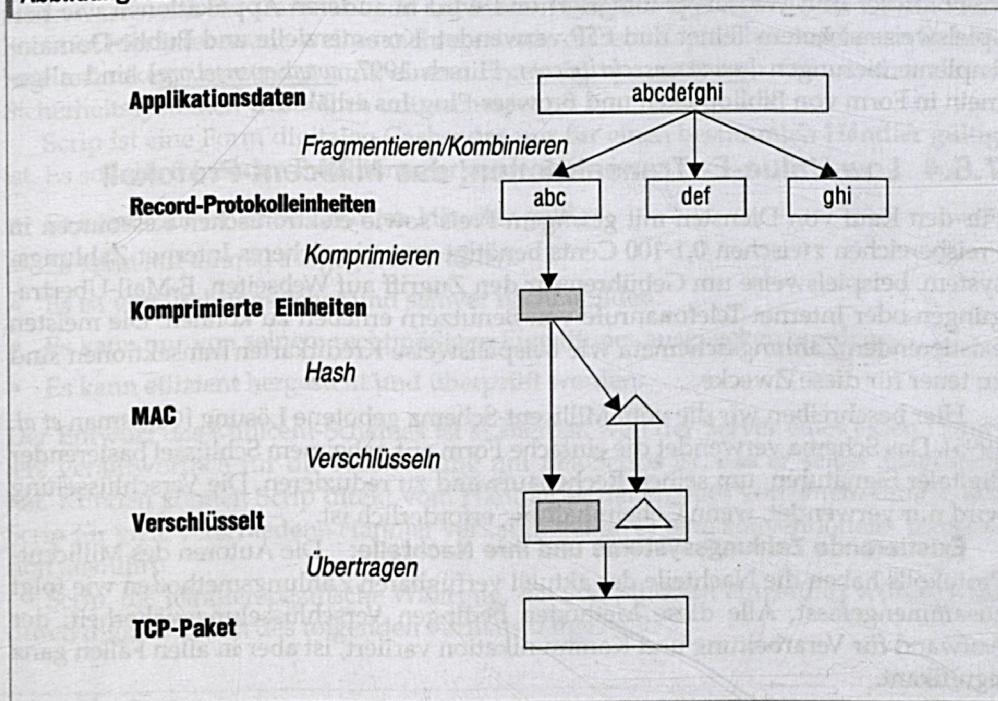
**Abbildung 7.20** SSL-Record-Protokoll

Abbildung 7.20 zeigt die Arbeitsweise des Record-Protokolls. Zunächst wird eine Nachricht für die Übertragung in Blöcke sinnvoller Größe zerlegt und anschließend werden die Blöcke optional komprimiert. Die Komprimierung ist nicht unbedingt ein Funktionsmerkmal der sicheren Kommunikation, aber sie wird hier unterstützt, weil ein Komprimierungs-Algorithmus einen Teil der Aufgaben bei der Verarbeitung der Datenmengen mit den Verschlüsselungs- und Authentifizierungs-Algorithmen gemeinsam erledigen kann. Mit anderen Worten, innerhalb der SSL-Record-Schicht kann eine Pipeline von Datentransformationen eingerichtet werden, die alle erforderlichen Transformationen effizienter erledigt, als sie unabhängig voneinander ausgeführt werden könnten.

Die Transformationen für Verschlüsselung und Nachrichtenauthentifizierung (MAC) wenden die in der vereinbarten Chiffrefolge angegebenen Algorithmen genau wie in den Abschnitten 7.3.1 und 7.4.2 beschrieben an. Schließlich wird der signierte

und verschlüsselte Block dem Partner über die zugeordnete TCP-Verbindung übermittelt, wobei die Transformationen umgekehrt werden, um den ursprünglichen Datenblock wiederherzustellen.

**Zusammenfassung** SSL stellt eine praktische Implementierung eines hybriden Verschlüsselungsschemas mit Authentifizierung und Schlüsselaustausch basierend auf öffentlichen Schlüsseln dar. Weil man sich im Handshake über die Chiffres einigt, ist man nicht von der Verfügbarkeit bestimmter Algorithmen abhängig. Und man ist auch nicht von sicheren Diensten für den Zeitpunkt der Sitzungseinrichtung abhängig. Die einzige Forderung bezieht sich auf Zertifikate mit öffentlichem Schlüssel, die von einer von beiden Parteien anerkannten Autorität stammen müssen.

Weil das Protokoll und eine Referenzimplementierung veröffentlicht wurden [Netscape 1996], wurde es ausgewertet und diskutiert. Frühe Entwürfe wurden verbessert und ein wichtiger Standard hat sich entwickelt. SSL ist heute in die meisten Webbrower und Webserver integriert und wird in anderen Applikationen wie beispielsweise sicherem Telnet und FTP verwendet. Kommerzielle und Public-Domain-Implementierungen [[www.rsasecurity.com](http://www.rsasecurity.com), Hirsch 1997, [www.openssl.org](http://www.openssl.org)] sind allgemein in Form von Bibliotheken und Browser-Plug-Ins erhältlich.

#### 7.6.4 Low-Value-E--Transaktionen: das Millicent-Protokoll

Für den Kauf von Diensten mit geringem Preis sowie elektronischen Ressourcen in Preisbereichen zwischen 0,1-100 Cents benötigt man ein sicheres Internet-Zahlungssystem, beispielsweise um Gebühren für den Zugriff auf Webseiten, E-Mail-Übertragungen oder Internet-Telefonanrufe von Benutzern erheben zu können. Die meisten existierenden Zahlungsschemata wie beispielsweise KreditkartenTransaktionen sind zu teuer für diese Zwecke.

Hier beschreiben wir die vom Millicent-Schema gebotene Lösung [Glassman *et al.* 1995]. Das Schema verwendet die einfache Form auf geheimem Schlüssel basierender digitaler Signaturen, um seinen Rechenaufwand zu reduzieren. Die Verschlüsselung wird nur verwendet, wenn Geheimhaltung erforderlich ist.

**Existierende Zahlungssysteme und ihre Nachteile:** Die Autoren des Millicent-Protokolls haben die Nachteile der aktuell verfügbaren Zahlungsmethoden wie folgt zusammengefasst. Alle diese Methoden bedingen Verschlüsselungssicherheit; der Aufwand für Verarbeitung und Kommunikation variiert, ist aber in allen Fällen ganz signifikant.

**Kreditkarten:** Das Problem bei der Verwendung von Kreditkarten für kleine Transaktionen sind die hohen Transaktionskosten, die entstehen, weil eine Kommunikation mit dem Zentralsystem des Kartenunternehmens erforderlich ist, und die weiter erhöht werden durch die zusätzlichen Kosten für die Sicherung elektronischer Transaktionen sowie durch andere Funktionen wie beispielsweise die Vorbereitung der Anweisungen für die Kunden.

**Kunden haben Konten bei Händlern:** Kunden müssen vor der ersten Transaktion ein Konto bei einem Händler einrichten. Die Transaktionskosten sind dann niedrig, aber der anfängliche Aufwand bewirkt, dass kleinere Transaktionen zunächst vermieden werden. Ein Händler muss einen Kundeneintrag in einer Kontendatenbank relativ lange aufbewahren.

**Transaktionen sammeln:** Wenn ein Kunde mehrere Käufe tätigt, kann der Händler sie aufzeichnen und anschließend in Rechnung stellen. Das ist der Kontenführung ähnlich, kann aber bestimmte Einrichtungskosten sparen. Der Händler muss die Auf-

zeichnungen auch für Kunden führen, die nur einen einzigen Kauf vornehmen und die entstehenden Kosten können den Gewinn übersteigen.

**Digitales Cash:** Wie das herkömmliche Geld sollte auch digitales Cash – d.h. digitale Wert-Token, die von einer Bank oder einem Händler ausgegeben werden – eine effiziente Möglichkeit bieten, kleine Transaktionen zu bezahlen, aber die Entwickler des digitalen Cash mussten das Problem einer *doppelten Ausgabe* lösen. Die doppelte Ausgabe ist eine Folge der Tatsache, dass der Inhaber eines digitalen Tokens möglicherweise mehrere nicht unterscheidbare Kopien erstellt. Token müssen also eindeutig identifizierbar sein und bei der Verwendung sofort überprüft und entwertet werden. Das Wichtigste dabei ist, ein Überprüfungsschema zu finden, das skalierbar, zuverlässig und in allen Situationen kosteneffektiv ist.

**Das Millicent-Schema** Das Millicent-Projekt entwickelte ein Schema für die sichere Verteilung und die Verwendung von *scrip* – eine spezielle Form elektronischen Cashs, das für Transaktionen mit kleinen Geldbeträgen entwickelt wurde. Millicent ist hier interessant, weil es mehrere der in diesem Kapitel beschriebenen Sicherheitstechniken verwendet, mit einem Ergebnis, das sich von SSL und anderen Sicherheitssystemen wesentlich unterscheidet.

Scrip ist eine Form digitalen Cashs, das nur für einen bestimmten Händler gültig ist. Es soll die folgenden Funktionsmerkmale aufweisen:

- Es ist nur für einen bestimmten Händler gültig.
- Es kann nur einmal ausgegeben werden.
- Es ist verfälschungssicher und schwer nachzubilden.
- Es kann nur von seinem rechtmäßigen Eigentümer ausgegeben werden.
- Es kann effizient hergestellt und überprüft werden.

Der Entwurf des Millicent-Schemas ist skalierbar, weil der Server eines jeden Händlers verantwortlich für die Auswertung nur des Scrips ist, das er selbst ausgestellt hat. Kunden können Scrip direkt vom Händler beziehen oder von einem Broker, der Scrip für viele verschiedene Händler verkauft, indem er eine konventionelle Transaktion ausführt.

Scrip, die händlerspezifische Währung, die von Millicent eingeführt wurde, wird durch digitale Token des folgenden Formats dargestellt:

| Händler | Wert | Scrip-ID | Kunden-ID | Ablauf-datum | Eigen-schaften | Zertifikat |
|---------|------|----------|-----------|--------------|----------------|------------|
|---------|------|----------|-----------|--------------|----------------|------------|

Das Feld *Eigenschaften* steht für vom Händler festgelegte Verwendungszwecke zur Verfügung – beispielsweise könnte es das Land oder den Staat enthalten, in dem der Kunde wohnt, sodass eine entsprechende Steuer ausgerechnet werden kann. Das *Zertifikat* ist eine digitale Signatur, die alle Felder im Scrip gegen Veränderung schützt. Die Signatur wird nach der in Abschnitt 7.4.2 beschriebenen MAC-Methode erzeugt. Die Aufgabe der restlichen Felder wird in der nachfolgenden Beschreibung deutlich.

Scrip wird von *Brokern* erzeugt und verteilt – Broker sind Server, die große Mengen Scrip verarbeiten und den Kunden und Händlern damit einen Teil des erforderlichen Aufwands abnehmen. Broker kaufen Scrip für reales Geld, kaufen Scrip (oder das Recht, es zu erzeugen) von Händlern billiger ein und verkaufen es gegen Kreditkarten- oder andere Zahlung an Kunden. Die Kunden können Scrip für mehr-