Co-funded by the Erasmus+ Programme of the European Union

InCyT-Trainingsmodul

Informationssicherheit für Mitarbeiter Einheit 2 Woche 3 Name der Einheit: Einführung in die Kryptographie	
Lernaktivitäten	**Webinar Übungen Workshop Präsentation Sonstiges**
Lehrverantwortliche	Claudio Fornaro
Ziele der Lernaktivitäten	1. Kryptographie 2. Kryptoanalyse 3. Algorithmus für einmaliges Auffüllen 4. Symmetrische Algorithmen 5. Asymmetrische Algorithmen 6. Hash-Funktionen 7. Signaturen und Public-Key-Infrastrukturen 8. Kryptographische Token 9. Challenge-Response-Mechanismus
Zu erwerbende Fertigkeiten	• Grundkenntnisse über Kryptographiealgorithmen und -infrastrukturen Technische Fertigkeit 2 - TS2
Dauer der Lernaktivität	1 Woche
Lernanforderungen	***Was wird benötigt?***
	• Es sind keine besonderen Vorkenntnisse erforderlich

Co-funded by the
Erasmus+ Programme
of the European Union

Kurze Informationen zum Modul

Die Kryptografie befasst sich mit Methoden zum Verbergen von Informationen vor der Offenlegung, sowohl während einer Kommunikation als auch zu Archivierungszwecken. Ihre Ziele werden durch den Entwurf und die Implementierung von Protokollen erreicht, die verhindern, dass ein unbefugter Dritter die Informationen abrufen kann. Bei einer sicheren Kommunikation kann der gegnerische Dritte nicht auf die übermittelten Informationen zugreifen, im Falle der Archivierung kann der Dritte den Inhalt des Archivierten nicht zurückverfolgen.

In der Kryptografie ist die dritte Partei immer eine böswillige Einheit, die darauf abzielt, Informationen abzurufen, zu denen sie keinen Zugang haben darf, sei es aus Gründen der Privatsphäre, des Geschäftsgeheimnisses oder der nationalen Sicherheit. Die Vertraulichkeit und Integrität der Daten, die Authentifizierung der beteiligten Parteien und die Nichtabstreitbarkeit dessen, was die Datenquelle übermittelt oder gespeichert hat, sind die Grundprinzipien der modernen Kryptografie.

Nach einem einführenden Teil, in dem die grundlegenden Konzepte beschrieben werden, stellen wir eine Reihe grundlegender kryptographischer Techniken vor, beginnend mit dem *One Time Pad Algorithmus* und dann aufbauend, um die beiden kryptographischen Algorithmenfamilien zu betrachten: Symmetrische Algorithmen und asymmetrische (oder Public Key) Algorithmen mit einem Vergleich zwischen ihnen. Die Rolle der Hash-Funktionen wird ebenfalls diskutiert, um digitale Signaturen einzuführen. Eine Beschreibung des Schlüsselkonzepts der Public-Key-Infrastrukturen wird ausführlich mit einigen Details zu den Standardmechanismen und -formaten vorgestellt. Abschließend werden einige Informationen über kryptografische Token und den Challenge-Response-Mechanismus gegeben.

Inhalt des Lernmaterials

Einführung

Kryptografie ist der Mechanismus, mit dem der Inhalt einer Nachricht vor anderen Entitäten verborgen wird. Mit anderen Worten geht es bei der Verschlüsselung darum, Nachrichten zu verschlüsseln, d. h. einen Klartext mit Hilfe eines Algorithmus und im Allgemeinen eines Schlüssels so zu verändern, dass der Inhalt der Nachricht unverständlich wird.

Die Kryptoanalyse untersucht, wie man die Nachricht entschlüsseln kann, ohne den Schlüssel zu kennen. Bei einem Angriff besteht die Kryptoanalyse in dem Versuch, den Inhalt der Nachricht zu ermitteln, ohne den Schlüssel zu kennen. Kryptographen und Kryptoanalytiker sind eigentlich ein und dieselbe Person: Wer die kryptographischen Algorithmen schreibt, analysiert sie auch, um ihre Schwachstellen zu finden oder ihre Robustheit zu überprüfen. Wie immer, wenn es um Sicherheit geht, ist die Vorstellung, Schutzprobleme lösen zu können, ohne zu wissen, wie Angriffstechniken funktionieren, nicht sehr effektiv.

Informationen in Nachrichten sind in der Regel redundant, d. h. in einem formalen Sinne verwenden Nachrichten in der Regel eine viel höhere Anzahl von Bits, als für die Übermittlung der Informationen tatsächlich erforderlich ist. **Die Redundanz von Informationen erleichtert die Kryptoanalyse**: Redundanz in den Informationen bedeutet, dass die Nachrichten eine Struktur aufweisen können, die es einfacher macht, den Inhalt der Nachricht zurückzuverfolgen, da nicht alle Nachrichten gleich wahrscheinlich sind. Wenn wir uns einen Text vorstellen, der mit normalen ASCII-Zeichen in Bytes dargestellt wird, wissen wir, dass die Kombinationen von Bits, die sinnvolle Nachrichten darstellen können, im Vergleich zu allen möglichen Kombinationen relativ gering sind. Die echten Nachrichten, die mit einer bestimmten Anzahl von Bytes dargestellt werden können, sind sehr gering. Dadurch ist es möglich, gültige Nachrichten zu identifizieren, indem man diejenigen verwirft, die keine echten Informationen enthalten. All dies erleichtert die Arbeit des Kryptoanalytikers ungemein, so dass der Zweck des Verschlüsselungsalgorithmus auch darin besteht, diese Redundanz zu verbergen, um die Informationen besser zu verbergen.

Beispiel: Kodierung *einer Folge* der vier Farben BLAU, GRÜN, ROT und GELB.

Diese können kodiert werden:

- mit 2 Bits (00, 01, 10, 11)
- als 4 verschiedene Großbuchstaben
- die Verschlüsselung der Farbnamen

Im ersten Fall wird die minimale Anzahl von Einheiten (Bits) verwendet, indem jeder Farbname durch die entsprechenden 2 Bits ersetzt wird, wodurch die ursprüngliche Sequenz verborgen wird, aber nicht die Reihenfolge, da es eine Entsprechung zwischen FARBE und den 2 Bits gibt. Im zweiten Fall, da jeder Buchstabe 8 Bits im ASCII-Code verwendet, sind 6 Bits für jede Farbe

überflüssig 75 % Redundanz und wir erreichen den gleichen Grad der Verschleierung wie mit den 2 Bits. Wir können Farben und Werte immer noch nicht miteinander in Verbindung bringen.

Betrachten Sie die folgende sehr einfache Chiffre, die so genannte *Caesar-Chiffre*, die nichts anderes tut, als die Buchstaben im Alphabet um eine bestimmte Anzahl von Positionen zu verschieben. Wenn wir sie zum Beispiel um 3 verschieben, werden alle A's zu D's, alle B's zu E's, und so weiter. Dies ist eine sehr einfache Chiffre, die eigentlich Julius Cäsar zugeschrieben wird, aber wenn die Farbnamen mit dieser Methode verschlüsselt werden, ist das sich wiederholende Muster ihrer verschlüsselten Buchstaben leicht zu erkennen (der ASCII-Code für R, gefolgt vom ASCII-Code für E, gefolgt vom ASCII-Code für D sind leicht als Einheit zu erkennen). Die Sequenz kann also identifiziert werden, und wenn die Wörter auch in diesem Fall bekannt sind, reicht es aus, die Anzahl der Buchstaben zu zählen, um die ursprünglichen Wörter zu identifizieren.

Anhand dieses einfachen Beispiels wird deutlich, dass die Kryptoanalyse keine vom Inhalt der Nachricht unabhängige Analyse ist: Eine völlig willkürliche Nachricht wäre viel schwieriger zu identifizieren und zu analysieren als eine Nachricht, die eine erkennbare Struktur hat.

Einige Arten von kryptografischen Angriffen und Algorithmen beruhen auf der Kenntnis eines Teils des Textes. In einigen Fällen ist es durch die Kenntnis eines Teils des Textes möglich, den Schlüssel zurückzuverfolgen und dann auf den Rest des Textes zuzugreifen. Dies ist bei der Netzkommunikation von großer Bedeutung, da bei der Netzkommunikation die Protokolle auf den verschiedenen Ebenen in der Regel eine bestimmte Menge an Informationen enthalten, die in jedem Fall erkennbar sind, z. B. wissen wir bei einem IP-Paket, dass der Header die Absender- und die Empfängeradresse enthält.

Ein Verschlüsselungsalgorithmus verbirgt auch redundante Informationen, so dass die Anwendung eines Komprimierungsalgorithmus auf eine verschlüsselte Nachricht keinen großen Vorteil bringt, da dies die Komprimierungsrate verringern würde. Wenn wir Komprimierung und Verschlüsselung verwenden wollen, müssen wir zuerst die Komprimierung und dann die Verschlüsselung verwenden.

Der One-Time-Pad-Algorithmus

Das One Time Pad (OTP) ist ein Algorithmus, der sozusagen "perfekt" ist. Es ist ein sehr einfacher Verschlüsselungsalgorithmus, sehr einfach zu implementieren und hat eine Eigenschaft, die ihn besonders interessant macht: Es ist bewiesen, dass es ohne Kenntnis des Schlüssels unmöglich ist, zum ursprünglichen Text zurückzukehren. Wir sehen uns diesen Algorithmus an und diskutieren dann, warum wir, da wir einen so guten Algorithmus haben, andere finden müssen.

Die EXKLUSIV-ODER- ⊕ (XOR) ist eine logische Funktion, bei der das Ergebnis wahr ist, wenn die beiden Operanden unterschiedlich sind. Sie arbeitet mit logischen Werten, die jedoch in binäre Ziffern übersetzt werden, in der Regel Wahr=1 und Falsch=0, ihre *Wahrheitstabelle* lautet:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

Aus dieser Tabelle ist ersichtlich, dass wenn $X \oplus Y \rightarrow Z$, dann $Z \oplus Y \rightarrow X$ und $Z \oplus X \rightarrow Y$. Das EXKLUSIV-ODER ist eine sehr einfache Funktion und auch leicht als elektronische Schaltung zu realisieren.

Der OTP-Mechanismus wird zur Übermittlung einer Nachricht zwischen zwei Parteien verwendet. In der wissenschaftlichen Literatur ist es üblich, die Akteure mit Personennamen zu identifizieren, sie werden gewöhnlich Alice (A) und Bob (B) genannt. Wir werden also beschreiben, wie Alice eine n-Byte-Nachricht (*PlainText*) sicher an Bob übertragen kann.

Der One-Time-Pad-Algorithmus erfordert eine Einrichtungsphase:

- Es wird eine Zufallsfolge von genau *n* Bytes generiert, das ist das *One Time Pad* (OTP) oder einfach der *Schlüssel*;
- Das OTP wird auf irgendeine Weise sicher zwischen den beiden Teilen ausgetauscht (im Voraus gegeben, über einen bereits gesicherten Kanal gesendet usw.).

Wenn beide Parteien im Besitz des Schlüssels sind, können sie ihn zur Übermittlung von Nachrichten verwenden. Falls Alice eine Nachricht an Bob senden möchte:

- Alice berechnet eine *bitweise* EXKLUSIVE ODER-Verknüpfung des *Klartextes* mit dem *Schlüssel*, wodurch ein *Chiffriertext* entsteht:
  Verschlüsselung: PlainText Schlüssel CypherText
- Der resultierende *CypherText* wird von Alice an Bob über einen möglicherweise unsicheren Kanal übermittelt
- Bob berechnet *Bit für Bit* das EXKLUSIVE ODER des *CypherTextes* und des *Schlüssels* und erhält so den ursprünglichen *Klartext* zurück:
  Entschlüsselung: CypherText Schlüssel PlainText

Beispiel

Nehmen wir an, Alice und Bob haben sich bereits auf einen zufälligen OTP (*Schlüssel*) geeinigt.

Alice möchte "HELLO" an Bob senden (in ASCII-Code, Leerzeichen sind hier der besseren Lesbarkeit halber eingefügt):

	H	**E**	**L**	**L**	**O**
Klartext:	01001000	01000101	01001100	01001100	01001111
			⊕		
Schlüssel:	10101001	01110010	10110010	10000110	11000110
			=		
Zifferntext:	11100001	00110111	11111110	11001010	10001001

Bob erhält den *CypherText* und wendet den *Schlüssel* an:

Zifferntext:	11100001	00110111	11111110	11001010	10001001
			⊕		
Schlüssel:	10101001	01110010	10110010	10000110	11000110
			=		
Klarxt:	01001000	01000101	01001100	01001100	01001111
	H	**E**	**L**	**L**	**O**

Der *PlainText* wurde wiederhergestellt.

Da der *Schlüssel* eine Zufallsfolge ist, kann man nicht sagen, ob ein bestimmtes Bit der Folge "0" oder "1" ist, da beide gleich wahrscheinlich sind. Ohne den *Schlüssel* haben wir bei einem CypherText-Bit keine Chance zu erraten, was das ursprüngliche Bit sein könnte. Dies gilt für einzelne Bits, für die gesamte Nachricht und für jedes Fragment unserer Nachricht.

Wenn ein Teil des *Schlüssels* bekannt ist, ist es möglich, den entsprechenden Teil des *Klartextes* wiederherzustellen, aber ohne den Rest des *Schlüssels* fehlt es nicht an Informationen über die anderen Bits des *Klartextes*. Daher ist die zuvor erwähnte Kryptoanalyse nicht wirksam. In diesem Fall wird der Rest des *Schlüssels* benötigt, um den verbleibenden Teil der ursprünglichen Nachricht zu entdecken. Alle vom *Schlüssel* übermittelten Informationen werden benötigt, um alle Informationen des Textes wiederherzustellen.

Einmalige Pad-Probleme

Die Stärke des OTP-Mechanismus beruht darauf, dass die Länge des *Schlüssels* genau der Länge des Textes entspricht und dass er nur einmal verwendet wird. Der größte Nachteil ist, dass die Übertragung des OTP die gleichen Probleme wie die der Nachricht hat.

In einigen Fällen lässt sich dieses Problem leicht lösen, z. B. könnte einer Botschaft eine Diplomatenmappe mit einem Band übergeben werden, das einen sehr langen Schlüssel enthält, und anschließend können die Nachrichten, die mit dieser Botschaft ausgetauscht werden, verschlüsselt werden, indem die Bits des Schlüssels nach und nach verbraucht werden. Das sind sehr eigenartige Anwendungen, aber sehr effektiv. Betrachtet man die Netzwerkkommunikation, so wird deutlich, dass die tatsächlichen Einsatzmöglichkeiten des One Time Pad sehr begrenzt sind. Es wird etwas Flexibleres benötigt, etwas, das den Austausch eines Schlüssels ermöglicht, der über einen sehr langen Zeitraum verwendet werden kann.

Betrachten wir jedes der Probleme.

Offline-Übertragung im Voraus

Dies ist ein allgemeines Problem mit Schlüsseln: Schlüssel müssen von den Parteien in irgendeiner Weise vereinbart werden. Diese Notwendigkeit, sich auf Schlüssel zu einigen, ist das wichtigste Problem bei der Verwendung von Kryptographie. Das Problem bei der Kryptographie sind nicht so sehr die Algorithmen und die Kryptoanalyse, sondern die Schlüsselverwaltung, d. h. die Gewissheit, dass alle und nur die richtigen Personen Zugang zu den Schlüsseln haben, der Austausch der Schlüssel im Voraus und die Gewissheit, dass die ausgetauschten Schlüssel die richtigen sind.

Einmalige Verwendung

Das One Time Pad wird, wie der Name schon sagt, nur einmal verwendet, was ein wichtiger Aspekt und eine Einschränkung ist. Wenn wir versuchen würden, denselben Schlüssel für zwei Nachrichten zu verwenden, würden wir die Sicherheit der Methode schwächen.

Die Erzeugung von Zufallszahlen in einem deterministischen System ist sehr schwierig

Bei Computern für allgemeine Zwecke werden Pseudozufallszahlen generiert, bei denen jede Zahl aus der vorhergehenden mit einer gewissen Rechenleistung errechnet wird, wodurch eine Folge von Zahlen entsteht, die sich nach einem sehr langen Zeitraum wiederholt. Dies sind keine echten Zufallszahlen, da ein Computer eine deterministische Maschine ist. In der Kryptographie ist diese Eigenschaft jedoch nachteilig. Wenn es möglich ist, die nächste Zufallszahl aus einer früheren abzuleiten, wird der Schutz, den kryptografische Algorithmen oder Protokolle bieten, wesentlich unsicherer. Das Schreiben von Code, der kryptografische Algorithmen implementiert, ist nicht trivial: Es gibt viele Details, auf die man achten muss, all die Fehler, die im Laufe der Jahre von Code-Entwicklern festgestellt wurden und die nur die Code-Entwickler in diesem speziellen Bereich zu

beherrschen gelernt haben. Das bedeutet, dass es immer eine schlechte Idee ist, eigene kryptografische Algorithmen zu entwickeln und/oder zu implementieren, anstatt gut getestete, weit verbreitete und umfassend validierte Bibliotheken zu verwenden.

Andere Algorithmen als OTP

Die anderen Algorithmen verwenden Schlüssel fester Länge. Wenn wir einen n-Bit-Schlüssel verwenden, haben wir 2^n verschiedene Schlüssel. Wenn der Wert von *n* groß genug ist und man einfach einen Brute-Force-Angriff durchführt (d. h. alle möglichen Schlüssel ausprobiert), würde es so viele Jahre dauern, den richtigen zu finden, dass "wahrscheinlich" der Inhalt der Nachricht nicht mehr von Interesse ist. Mit einem der schnellsten verfügbaren Supercomputer würde die Überprüfung eines 256-Bit-Schlüssels beispielsweise $3{,}67\times10^{55}$ Jahre dauern. Es ist klar, dass die Verwendung von Schlüsseln mit ausreichender Länge selbst für diejenigen, die rechtmäßig ver- und entschlüsseln, zusätzliche Rechenkosten verursacht, aber die Auswirkungen sind begrenzt.

Es gibt symmetrische und asymmetrische Algorithmen.

Symmetrische Algorithmen

Die typischste und älteste Familie kryptografischer Algorithmen, an die wir in der Regel denken, wenn wir an die Verschlüsselung von Nachrichten denken, sind die ***symmetrischen Algorithmen***, benannt nach der Symmetrie der Ver- und Entschlüsselungsvorgänge. Diese Algorithmen verwenden einen einzigen Schlüssel, der sowohl dem Absender als auch dem Empfänger bekannt sein muss; er wird zur Ver- und Entschlüsselung einer Nachricht verwendet.

Ein einfaches symmetrisches Schema könnte dasselbe sein wie OTP, da OTP ein symmetrischer Algorithmus ist, mit dem großen Unterschied, dass der *Schlüssel* eine feste Länge hat (z. B. 256 Bits) und in irgendeiner Weise wiederholt wird, möglicherweise nach einer komplexen Änderung jedes Mal:

Verschlüsselung: *encryption_function*(PlainText, Key) CypherText

Entschlüsselung: *decryption_function*(CypherText, Key) PlainText

Allgemeiner ausgedrückt: Selbst wenn wir denselben *Schlüssel* verwenden, ist die zur Entschlüsselung verwendete Funktion nicht unbedingt dieselbe, die zur Verschlüsselung verwendet wird. Betrachten wir zum Beispiel die Caesar-Chiffre, so ist es klar, dass die Funktion, die die Nachricht entschlüsselt, das Alphabet nach links und nicht nach rechts verschiebt, aber die Entschlüsselung verwendet denselben Schlüssel: die Nummer 3.

Einige wichtige symmetrische Schlüsselalgorithmen

DES (Datenverschlüsselungsstandard)

Dieser historische, inzwischen überholte Algorithmus verwendet einen 56-Bit-Schlüssel, der in den späten 1970er Jahren (als der Algorithmus entwickelt wurde) für viele Anwendungen absolut ausreichend war, da ein Brute-Force-Angriff auf einen 56-Bit-Schlüssel, um die Nachricht mit den damals verfügbaren Rechenressourcen und in den Kontexten, in denen dieser Algorithmus verwendet wurde, zu erhalten, unpraktisch war. 40 Jahre später würde der DES-Algorithmus nicht einmal ein paar Minuten auf einem Heimcomputer überleben. Dies ist ein sehr wichtiger Aspekt: Die Länge des Schlüssels wird auf der Grundlage der *tatsächlich* verfügbaren Rechenkapazitäten gewählt. Ein Schlüssel, der heute als robust genug angesehen wird, kann in 20 Jahren nicht mehr so stark sein. Im Allgemeinen neigen wir dazu, die Länge des Schlüssels im Überfluss zu wählen, um sicherzustellen, dass ein gewisser Spielraum vorhanden ist, sowohl weil die Rechenkapazitäten zunehmen, als auch weil die Kryptoanalyse als Disziplin Fortschritte macht und möglicherweise Schwachstellen im Algorithmus finden kann. Dies ist ein recht interessanter Aspekt. Anders als man meinen könnte, versagt ein kryptografischer Algorithmus normalerweise nicht plötzlich, d. h. an dem Tag, an dem eine Schwachstelle in einem Algorithmus entdeckt wird, wird er sofort von sicher zu völlig unsicher und jeder kann den Klartext im Handumdrehen finden. Was passiert, ist, dass Forscher den Algorithmus analysieren und vielleicht kleine Schwachstellen finden, die in bestimmten Fällen einen Angriff erleichtern können. Dann findet jemand möglicherweise eine Optimierung des Algorithmus, um das Problem zu lösen. Im Laufe der Zeit wird der Algorithmus aufgrund unlösbarer Probleme allmählich geschwächt, bis er für bestimmte Anwendungen oder überhaupt nicht mehr geeignet ist. Es gibt eine fortschreitende Erosion der Verteidigungsfähigkeit kryptographischer Algorithmen, und diese fortschreitende Schwächung ist ein schwieriger zu handhabendes Problem, weil wir natürlich nicht wissen können, welche Fortschritte es in der Zukunft aus der Sicht der Kryptoanalyse geben wird.

AES (Advanced Encryption Standard, auch bekannt unter seinem ursprünglichen Namen Rijndael)

AES wurde nicht nur aufgrund seiner Robustheit, sondern auch aufgrund seiner effizienten Implementierung in typischen Architekturen ausgewählt. Es wurde kein geheimer Algorithmus gesucht, denn ein geheimer Algorithmus wird nicht als robuster angesehen als ein nicht geheimer: Im Gegenteil, der Algorithmus sollte von möglichst vielen wertvollen Kryptoanalytikern analysiert werden, um sicherzustellen, dass er von sich aus robust ist. Ein Algorithmus, der zwar sicher, aber zu langsam und zu teuer in Bezug auf die Rechenressourcen ist, wäre nicht verwendbar. Diese Einschränkung war für die Wettbewerber am schwierigsten zu überwinden, verglichen mit der der Robustheit.

Asymmetrische (oder Public Key) Algorithmen

Asymmetrische Algorithmen verwenden nicht nur einen Schlüssel, sondern zwei: ein Schlüssel wird zur Verschlüsselung und der andere zur Entschlüsselung verwendet. Daher verwenden Sender und Empfänger beim Austausch von Nachrichten nicht denselben Schlüssel. Diese Schlüssel werden

paarweise erzeugt und verwaltet: Was ein Schlüssel (irgendeiner des Paares) verschlüsselt, kann nur mit dem anderen Schlüssel des Paares entschlüsselt werden.

Der Grund für die Verwendung asymmetrischer Algorithmen ergibt sich aus den Problemen der symmetrischen Kryptografie:

1. Eine vertrauliche Kommunikation zwischen zwei Einheiten erfordert, dass nur ein einziger Schlüssel zwischen ihnen ausgetauscht wird. Wenn die Einheiten *n* sind, sind $n \cdot (n-1)/2$ verschiedene Schlüssel erforderlich, um eine vertrauliche Kommunikation zwischen jedem Paar der Parteien zu ermöglichen.
2. Sowohl der Sender als auch der Empfänger kennen ihren gemeinsamen Schlüssel, so dass *eine Nichtabstreitbarkeit* (Abstreitbarkeit ist das Bestreiten, der Autor einer bestimmten Nachricht zu sein) nicht möglich ist.

Das erste Problem ist nicht die Anzahl der Schlüssel, die jedes Subjekt verwalten muss, sondern der *Austausch* dieser Schlüssel. Jedes Subjekt muss sich mit jedem der anderen n-1 Subjekte auf einen Schlüssel einigen, was in der Tat eine komplexe Operation ist; dies macht die symmetrische Kryptographie für viele Anwendungen unpraktisch.

Das zweite Problem, die *Nicht-Abstreitbarkeit*, liegt darin, dass wir oft sicher sein wollen, dass eine bestimmte Nachricht tatsächlich von einem bestimmten Subjekt gesendet wurde. Wenn ein Subjekt eine Nachricht von einem anderen Subjekt erhält und die beiden Subjekte die einzigen sind, die den zur Verschlüsselung verwendeten Schlüssel kennen, ist jedes Subjekt sicher, dass das andere Subjekt der Autor der Nachricht ist. Umgekehrt weiß jedes Subjekt, da es nur den Schlüssel kennt, dass nur das andere Subjekt die Nachricht lesen kann. Das Problem tritt auf, wenn dies gegenüber einem Dritten nachgewiesen werden muss, z. B. bei einem Rechtsstreit, da dieser Mechanismus nicht funktioniert: Jede Partei kann eine Nachricht erstellen, sie verschlüsseln und dann behaupten, dass die andere Partei der Verfasser ist. Das Problem der *Nichtabstreitbarkeit* ist mit symmetrischer Kryptographie nicht lösbar.

Das asymmetrische Schema ähnelt dem symmetrischen, aber die verwendeten Schlüssel sind die beiden, die paarweise erzeugt werden:

Verschlüsselung: *encryption_function*(*PlainText*, ***Key1***) *CypherText*

Entschlüsselung: *decryption_function*(*CypherText*, ***Key2***) *PlainText*

Die Namen ***Key1*** und ***Key2 wurden*** anstelle von *EncryptionKey* und *DecryptionKey* verwendet, weil die beiden Schlüssel mathematisch austauschbar sind. Das Subjekt, das die Schlüssel generiert, hält in der Regel einen privat/geheim (den so genannten *privaten Schlüssel*) und macht den anderen öffentlich (den so genannten *öffentlichen Schlüssel*), daher der Name der Algorithmusklasse, die

auch als *Public-Key-Algorithmus bezeichnet wird*. Auf den Begriff "*öffentlich*" wird später noch näher eingegangen.

Die zum Verschlüsseln verwendete Funktion ist nicht notwendigerweise die gleiche wie die zum Entschlüsseln verwendete. Wichtig ist, dass sie zusammen den kryptografischen Algorithmus darstellen, wobei ein Teil auf dem einen Schlüssel und der andere Teil auf dem anderen Schlüssel beruht.

Die beiden Schlüssel haben solche mathematischen Eigenschaften, dass die Ermittlung des einen Schlüssels aus dem anderen sehr komplex ist und einen enormen Zeitaufwand erfordert. Insbesondere würde die Rekonstruktion des privaten Schlüssels selbst bei Kenntnis des öffentlichen Schlüssels, des *Klartextes* und des *Chiffriertextes* eine unangemessen lange Zeit in Anspruch nehmen.

Der asymmetrische Algorithmus verwendet

Angenommen, der *öffentliche Schlüssel* ist öffentlich zugänglich und sein Besitzer ist der einzige, der den zugehörigen privaten Schlüssel kennt (Einzelheiten dazu später). Da der öffentliche Schlüssel für jedermann zugänglich ist, kann jeder ihn zur Verschlüsselung einer Nachricht verwenden, aber nur der Besitzer des entsprechenden privaten Schlüssels kann die Nachricht entschlüsseln, was die **Vertraulichkeit** der Nachrichtenübertragung gewährleistet.

Wir sind also von einer Situation, in der ein Subjekt *n* verschiedene Schlüssel benötigte, um vertraulich mit *n* anderen Subjekten zu kommunizieren, zu einer Situation übergegangen, in der nur noch ein Schlüssel (der öffentliche Schlüssel des Empfängers) benötigt wird, damit jeder eine vertrauliche Nachricht an ein Subjekt senden kann.

Damit *n* Subjekte miteinander kommunizieren können, muss jedes Subjekt ein Schlüsselpaar besitzen, so dass die Gesamtzahl der Schlüssel 2·n beträgt. Das Interessante daran ist nicht so sehr, dass die Anzahl der Schlüssel abgenommen hat, sondern dass diese als öffentliche Schlüssel auf viel einfachere Weise kommuniziert werden können. Wenn jemand anderes von diesen öffentlichen Schlüsseln erfährt, ist das überhaupt kein Problem. Daher können diese Schlüssel wirklich veröffentlicht werden, jedem zugänglich gemacht werden und jeder kann sie verwenden, ohne den Schutz der Nachrichten zu schwächen.

Wenn das Subjekt der einzige Besitzer eines privaten (geheimen) Schlüssels ist, dessen öffentliches Gegenstück öffentlich zugänglich ist, kann jeder mit dem privaten Schlüssel des Subjekts verschlüsselte Nachrichten verifizieren (entschlüsseln). Wenn es also möglich ist, den öffentlichen Schlüssel eines Subjekts zur Entschlüsselung einer Nachricht zu verwenden, bedeutet dies, dass nur das Subjekt, das den entsprechenden privaten Schlüssel besitzt, die Nachricht verschlüsselt haben kann, was die **Nichtabstreitbarkeit der Nachricht** gewährleistet.

Ein Problem ist noch zu lösen: Es muss sichergestellt werden, dass ein öffentlicher Schlüssel wirklich mit dem rechtmäßigen Subjekt verbunden ist. Hier muss eine *überparteiliche* Zertifizierungsmethode zur Verfügung gestellt werden, die von einer *Public Key Infrastucture* (PKI) bereitgestellt wird (siehe unten).

Nebenbei bemerkt, können asymmetrische Algorithmen auch als *Challenge-Response-Algorithmus* verwendet werden. Damit Subjekt A Subjekt B *authentifizieren* (beweisen, dass ein Subjekt wirklich derjenige ist, der es zu sein behauptet), sendet A (im Klartext) eine Zufallszahl an B; dann verschlüsselt B sie mit seinem privaten Schlüssel und sendet sie zurück an A. Wenn A in der Lage ist, die Nachricht mit dem öffentlichen Schlüssel von B zu entschlüsseln, ist A sicher, dass die andere Partei B ist, da er der einzige ist, der etwas verschlüsseln kann, das der öffentliche Schlüssel von B entschlüsseln kann.

Vorteile und Benachteiligungen

Die Gesamtzahl der Schlüssel ist deutlich geringer, was die Komplexität der Gewährleistung der korrekten Verwaltung aller Schlüssel verringert (und nur der Eigentümer kennt den geheimen Schlüssel). Da die öffentlichen Schlüssel öffentlich zugänglich gemacht werden, sind keine besonderen Vereinbarungen zwischen verschiedenen Subjekten erforderlich. Diese Verwendung öffentlicher Schlüssel ist beispielsweise im elektronischen Geschäftsverkehr von großer Bedeutung, da der elektronische Geschäftsverkehr nicht voraussetzen kann, dass es eine vorherige Vereinbarung zwischen dem Verkäufer und dem Kunden gibt.

Um diese Verbindung zwischen den beiden Schlüsseln herzustellen und gleichzeitig zu gewährleisten, dass der eine nicht aus dem anderen berechnet werden kann und dass das, was mit dem einen verschlüsselt wird, mit dem anderen entschlüsselt werden kann, sind asymmetrische Algorithmen in Bezug auf die mathematischen Probleme viel komplexer als symmetrische Algorithmen. Diese mathematischen Probleme erfordern im Wesentlichen langsamere Algorithmen als symmetrische Algorithmen und die Verwendung von längeren Schlüsseln, zumindest bei den derzeit verwendeten Algorithmen. Daher sind sie zwar einerseits flexibler, andererseits aber auch aufwändiger in Bezug auf die Rechenressourcen.

Prominenter Algorithmus: RSA *(nach den Initialen von Ronald Rivest, Adi Shamir und Leonard Adleman)*
Dieser Algorithmus basiert auf der Komplexität der Primfaktorzerlegung: Es ist einfach, zwei Primzahlen zu multiplizieren und das Ergebnis zu erhalten, aber es ist komplex/aufwendig, die beiden Primzahlen aus dem Ergebnis zu ermitteln. Die Primfaktorzerlegung ist ein schwieriges Problem. Wir wissen auch, dass es nicht einfach ist, sehr hohe Primzahlen zu finden, da es keine Regel gibt, die es erlaubt, Primzahlen herauszufinden, außer zu versuchen, sie zu faktorisieren und herauszufinden, ob sie Primzahlen sind. Das bedeutet, dass bei sehr großen Zahlen die Ermittlung der beiden Primzahlen, von denen eine bestimmte Zahl abgeleitet ist, ein Problem darstellt, das

erhebliche Rechenressourcen erfordert. Andererseits können legitime Operationen mit begrenzten Rechenressourcen durchgeführt werden.

Prominenter Algorithmus: Elliptische Kryptographie *(ECC - Elliptic Curve Cryptography)*

Das "schwierige Problem" ist in diesem Fall die Berechnung des diskreten Logarithmus (in einem endlichen Feld). Ein "endliches Feld" bedeutet, dass die Werte ganzer Zahlen in einer begrenzten Menge enthalten sind (sehr grobe Definition).

Das ECC-Problem ist komplexer als die Primfaktorzerlegung, so dass Sie die gleiche Sicherheit mit kürzeren Schlüsseln erhalten. ECC-Algorithmen sind schneller als RSA.

Vergleich von symmetrischen und asymmetrischen Algorithmen

Asymmetrische Algorithmen nutzen andere mathematische Probleme als die der symmetrischen Kryptographie. Dies bedeutet, dass die Schlüssellängen nicht mit denen der symmetrischen Kryptographie vergleichbar sind. Das Problem bei der symmetrischen Verschlüsselung besteht lediglich darin, dass der Schlüssel lang genug ist, um zu verhindern, dass alle möglichen Zahlen, die zu einem Schlüssel passen könnten, ausprobiert werden. Bei einem 128-Bit-Schlüssel gibt es also 2^{128} (das sind $3{,}4 \cdot 10^{38}$) mögliche Schlüssel. Die heutigen symmetrischen Algorithmen verwenden in der Regel 128-, 256- und manchmal 512-Bit-Schlüssel.

Betrachtet man beispielsweise RSA, das sich Primzahlen zunutze macht, so ist klar, dass nicht alle Schlüssel möglich sind, da sie an das Vorhandensein von Primzahlen gebunden sind. Nicht alle Zahlen sind Primzahlen, also sind auch nicht alle Zahlen mögliche Schlüssel. Sehr vereinfacht ausgedrückt, benötigt der RSA-Algorithmus einen Schlüssel in der Größenordnung von 1024 Bit, um die gleiche Robustheit des Algorithmus (und damit die gleiche Schwierigkeit aus rechnerischer Sicht) wie ein symmetrischer 128-Bit-Algorithmus zu erreichen. Die heutigen asymmetrischen Algorithmen verwenden in der Regel 1024-, 2048- und 4096-Bit-Schlüssel.

Hash-Funktionen

Hash-Funktionen berechnen eine kurze Zusammenfassung eines Textes beliebiger Länge mit fester Länge; diese Zusammenfassung wird *Digest* genannt. Eine Hash-Funktion muss einfach und schnell zu berechnen sein.

Sicherheitseigenschaften

Hash-Funktionen sind "Einweg-Funktionen" in dem Sinne, dass es sehr komplex und langsam ist, aus dem Digest einen *möglichen* Text wiederherzustellen, der ihn erzeugt, aber dieser Text ist kaum brauchbar. Dies ist eine Folge der Tatsache, dass der ursprüngliche Text-Digest größer ist (und sein sollte) als der Digest, so dass praktisch unendliche Folgen von Bytes zu einem bestimmten Digest führen können. Selbst wenn ein Text mit dem angegebenen Digest gefunden wird, ist es äußerst

unwahrscheinlich, dass es sich um den Originaltext handelt oder dass er überhaupt einen Sinn ergibt. Es ist extrem schwierig, zwei sinnvolle Nachrichten mit demselben Digest zu erstellen oder eine sinnvolle Nachricht aus einem gegebenen Digest zu erstellen, da jede Änderung einer Nachricht, selbst ein einziges Bit, einen völlig anderen Digest ergibt (etwa 50 % der Bits sind unterschiedlich).

Integritätsprüfung

Bei einer Nachricht M wird ihr Digest D mit Hilfe einer Hash-Funktion H berechnet: D = H(M)

Eine sichere Übertragung des Digests D ermöglicht es, die Korrektheit der Nachricht M zu gewährleisten: Der Empfänger der Nachricht M berechnet den Digest und vergleicht ihn mit dem empfangenen Digest (D); sind beide identisch, wurde die Nachricht nicht verändert. Es ist wichtig, dass der Digest auf sichere Weise übertragen wird, da sonst kein Schutz gegeben ist.

Dieser Mechanismus wird auch zur Überprüfung einer heruntergeladenen Software anstelle anderer, weniger leistungsfähiger (aber schneller) Algorithmen verwendet, die als CRCs (Cyclic Redundancy Checks) bekannt sind und in Netzwerkprotokollen üblich sind.

Einige wichtige Hash-Algorithmen

Im Folgenden sind zwei der bekanntesten Hash-Algorithmen aufgeführt.

MD5 (Message Digest #5, 1991)

MD5 erfordert eine geringe Rechenleistung (im Vergleich zu den meisten anderen Hash-Algorithmen) und erzeugt einen 128-Bit-Hash. Im Laufe der Zeit wurde er allmählich geschwächt, so dass er für Sicherheitszwecke völlig unzuverlässig ist (derzeit ist es möglich, innerhalb von Sekunden zwei verschiedene Nachrichten mit demselben MD5-Hashwert zu finden - was als *Kollision* bezeichnet wird). Dennoch ist er für nicht kryptografische Anwendungen immer noch nützlich, zum Beispiel um Übertragungsfehler anstelle des gebräuchlicheren (und schwächeren, aber schnelleren) CRC-32 zu entdecken.

SHA-3 (Sicherer Hash-Algorithmus #3, 2015)

SHA ist eigentlich eine Familie von sicheren Hash-Algorithmen, SHA-3 ist die letzte Version und das Ergebnis eines Wettbewerbs unter Kryptographen. Der Algorithmus kann so eingestellt werden, dass ein Gleichgewicht zwischen Sicherheit und Geschwindigkeit erreicht wird; er erzeugt Hash-Werte von 224, 256, 334 und 512 Bit. Er ist 2-3 mal langsamer als MD5, aber bisher wurden keine Schwachstellen gefunden.

Signaturen (Digitale Signaturen)

Bei digitalen Signaturen, die in den Rechtsvorschriften einiger Länder als elektronische Signatur bezeichnet werden, wird eine Nachricht genommen, ein Hash berechnet und mit dem geheimen Schlüssel eines asymmetrischen Algorithmus verschlüsselt (nur der Hash). Dieser mit dem geheimen Schlüssel signierte Hash wird als *Signatur* bezeichnet. Eine digitale Signatur verschlüsselt also nicht die ursprüngliche Nachricht, sondern garantiert nur, dass der entsprechende Text nicht verändert wurde, da sich sonst der Digest ändern würde. Sie garantiert auch, dass der Ursprung verifiziert werden kann (*Nichtabstreitbarkeit*), da die Nachricht mit dem privaten Schlüssel eines asymmetrischen Algorithmus verschlüsselt wurde. Das bedeutet, dass jeder überprüfen kann, ob der Text intakt ist und tatsächlich von einem bestimmten Subjekt erzeugt wurde.

Ähnlich wie digitale Signaturen sind **Message Authentication Codes** (MACs). Auch sie sollen die Integrität und Authentizität einer Nachricht gewährleisten. Der Digest wird durch Hashing des zu schützenden Textes und eines *symmetrischen* Algorithmus-Schlüssels berechnet (es gibt keine Zusicherung der Nichtabstreitbarkeit, daher ist er nur geeignet, wenn diese Eigenschaft nicht benötigt wird). Einer der bekanntesten Algorithmen ist HMAC: Keyed-hash MAC.

Infrastrukturen für öffentliche Schlüssel

Die Anforderungen an Vertraulichkeit, Integrität und Authentizität asymmetrischer Verschlüsselungs- und digitaler Signaturverfahren erfordern den Aufbau eines Vertrauensverhältnisses zwischen den beteiligten Parteien. Da die Parteien oft vorher keinen Kontakt hatten, muss eine (von ihnen) vertrauenswürdige dritte Partei die erforderlichen Garantien bieten, d.h. diese dritte Partei muss die Authentizität des Besitzers eines öffentlichen Schlüssels sicherstellen.

Eine *Zertifizierungsstelle* (CA) ist Teil eines hierarchischen Vertrauensmodells, das die Identität des Benutzers garantiert. Sie muss von allen an der Kommunikation beteiligten Stellen als "vertrauenswürdig" anerkannt werden.

Um sich der Authentizität eines Subjekts zu versichern, unterzeichnet eine CA ein *digitales Zertifikat*, das das Subjekt mit seinem öffentlichen Schlüssel verknüpft (es ist konzeptionell ähnlich wie ein Identitätsdokument einer Person). Die von der CA angewendete Signatur ist eine digitale Signatur, d. h. sie wird mit dem privaten Schlüssel der CA errechnet. Natürlich muss das digitale Zertifikat mit dem öffentlichen Schlüssel der Zertifizierungsstelle validiert werden.

Eine *Infrastruktur für öffentliche Schlüssel* (PKI) ist eine hierarchische Struktur von Zertifizierungsstellen, bei der jede Zertifizierungsstelle ihren öffentlichen Schlüssel von der vorhergehenden Zertifizierungsstelle signieren lässt (die zu diesem Zweck ihren privaten Schlüssel

verwendet); der Prozess geht bis zur *Stammzertifizierungsstelle*, die ein Zertifikat mit ihrem öffentlichen Schlüssel unter Verwendung ihres privaten Schlüssels selbst signiert.

Ein hierarchisches Modell ermöglicht den Aufbau beliebig tiefer Hierarchien. Innerhalb eines Landes könnten wir beispielsweise die kommunalen, ..., regionalen Zertifizierungsstellen bis hin zu einer nationalen Stammzertifizierungsstelle haben.

Das Root-CA-Zertifikat muss an vielen Stellen und auf verschiedene Weise veröffentlicht werden, damit es nicht durch ein gefälschtes Zertifikat ersetzt werden kann. So werden beispielsweise die Betriebssysteme selbst mit einer Reihe von Stammzertifikaten und vertrauenswürdigen Zwischenzertifikaten ausgeliefert. Abbildung 1 zeigt einen Auszug aus der Liste der Root-Zertifikate eines MS Windows 10-Betriebssystems.

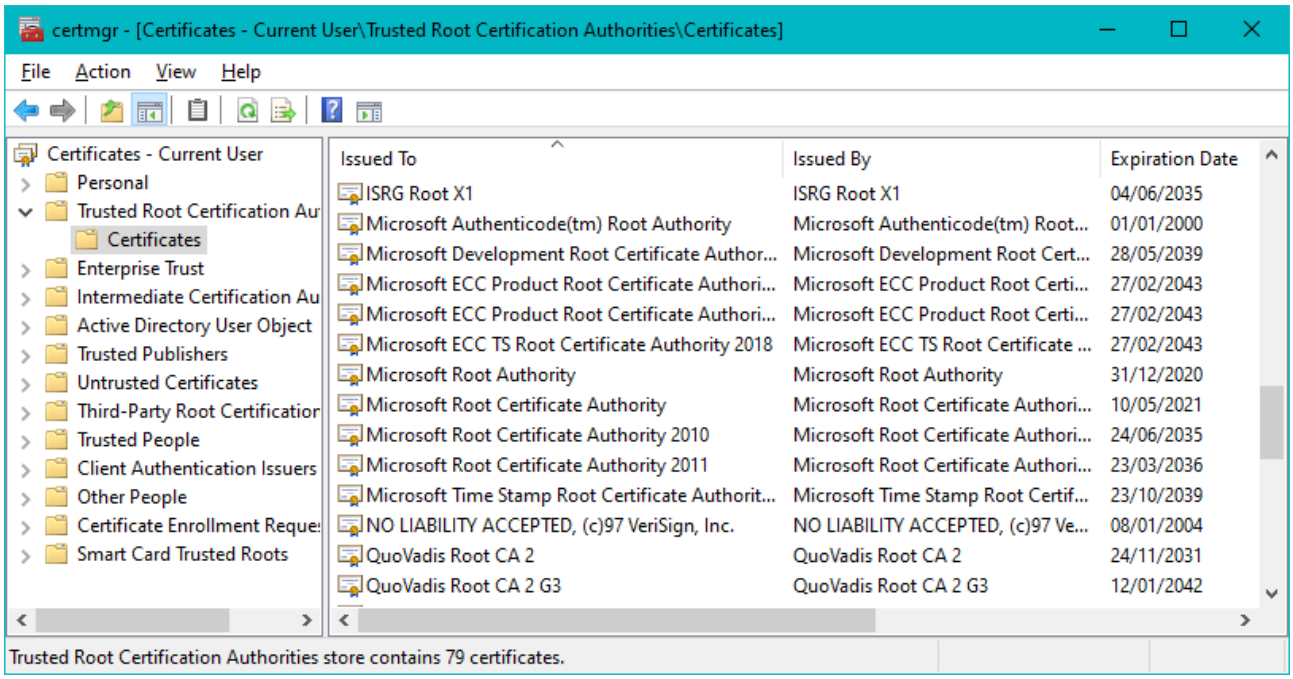

Abbildung 1. Windows 10 Zertifikat-Manager

Bestandteile einer PKI

Der Hauptakteur ist die **Zertifizierungsstelle (CA)**, die die kryptografischen Schlüssel und Zertifikate der PKI verwaltet. Sie ist die Komponente, die über die eigentlichen Schlüssel verfügt und dann die kryptografischen Operationen durchführt. In diesem Sinne ist sie natürlich auch das kritischste Element der Zertifizierungsstelle.

Die **Registration Authority (RA)** verwaltet Zertifikatsanträge und stellt ausgestellte Zertifikate bereit. Sie befasst sich mit der Verteilung von kryptografischen Token und Software (sie ist das Front-End der CA). Sie ist wie ein Schalter, zu dem ein Bürger gehen kann, um ein digitales Zertifikat anzufordern und von dem er es erhalten kann. Dabei kann es sich um einen physischen Schalter

oder eine Website handeln, wobei die Art des Schalters mit verschiedenen Sicherheitsniveaus verbunden ist. Darüber hinaus kann eine RA in der Regel auch den Nutzer bei der Erzeugung des Schlüsselpaars und der sicheren Verwaltung dieser Zertifikate unterstützen, indem sie beispielsweise Ad-hoc-Software oder Hardware-Tools bereitstellt, die den Antragsteller bei der Erzeugung von Signaturen, der Verschlüsselung von Dokumenten, der Überprüfung von Signaturen und der Entschlüsselung von Informationen unterstützen.

Ein **Zertifikatsserver** ist ein Verzeichnis, in dem Benutzerzertifikate veröffentlicht und auf irgendeine Weise durchsuchbar gemacht werden. Das Verzeichnis entspricht dem X.500-Standard und wird über das LDAP-Protokoll abgefragt.

Das digitale Zertifikat

Die Struktur und das Format digitaler Zertifikate werden in der ITU-T-Norm X.509 beschrieben (es handelt sich um eine Norm der Internationalen Fernmeldeunion mit der Nummer X.509).

Ein digitales X.509-Zertifikat muss mindestens die folgenden Felder enthalten:

DN (Unterscheidungsname)/distinguish name) Benutzer: Gaius Julius Cäsar

CN (allgemeiner Name/common name): Julius Caesar

O (Organisation): Regierung

OU (Organisationseinheit/organization unity): Senat

C (Land/country): Römisches Reich

Dies ist das klassische Muster eines Zertifikats, das dem X.500-Standard entspricht, zu dessen Familie auch X.509 gehört. Ein Zertifikat kann auch andere Attribute haben (z. B. E-Mail). Es kann eine IP-Adresse oder eine URL enthalten, die einem X.500-Verzeichnispfad entspricht (z. B. LDAP - Lightweight Directory Access Protocol):

IT / Römisches Reich / Regierung / Julius Caesar / Gaius Julius Caesar /

Ein digitales X.509-Zertifikat muss außerdem mindestens die folgenden Felder enthalten:

DN Issuer: Name der Behörde im X.500-Format (die Zertifizierungsbehörde hat auch ihren eigenen Distinguished Name, der im Zertifikat erscheint. Natürlich ist das Format, in dem er dargestellt wird, dasselbe wie der Distinguished Name des Subjekts, für das das Zertifikat gilt.

Seriennummer: Seriennummer des von der Behörde gehaltenen Zertifikats (jedes Zertifikat wird durch die ihm von der Behörde zugewiesene Seriennummer eindeutig identifiziert. Andererseits

identifiziert das Paar (*Seriennummer* & *Name des Ausstellers*) ein Zertifikat in der Fülle von Zertifikaten, die weltweit ausgestellt werden können, als eindeutiges Paar).

NotBefore: Datum der Aktivierung

NotAfter: Verfallsdatum

Dies sind zwei Informationen, die eng miteinander verbunden sind. Ein Zertifikat hat also ein Anfangsdatum und ein Verfallsdatum, nach dem es nicht mehr gültig ist. Der Grund dafür ist der Wunsch, die Schlüssel vor übermäßigem Gebrauch zu schützen.

Öffentlicher Schlüssel: der öffentliche Schlüssel der zertifizierten Person (das Zertifikat enthält persönliche Daten und den öffentlichen Schlüssel)

Algorithmus des öffentlichen Schlüssels: der Algorithmus, für den der öffentliche Schlüssel geeignet ist

Signaturalgorithmus: der von der Behörde verwendete Algorithmus zum Signieren des Zertifikats

Signatur: die von der Zertifizierungsstelle auf dem Zertifikat angebrachte Unterschrift (also das Element, das die Authentizität des Zertifikats selbst garantiert)

KeyUsage: die Zwecke, für die der Schlüssel verwendet werden kann

Erstellung und Widerruf von Zertifikaten

Der Standard RFC 2314 (aus PKCS # 10) definiert das Format der certificateRequest-Nachricht und enthält den öffentlichen Schlüssel und die Daten des Antragstellers (beides wird einer RA übergeben). Die Anforderungsnachricht erfordert den sogenannten *Nachweis des Besitzes* des privaten Schlüssels: Die certificateRequest-Nachricht muss mit dem privaten Schlüssel des Antragstellers signiert werden. Durch die Überprüfung dieser Signatur bestätigt die Behörde im Wesentlichen, dass der Antragsteller tatsächlich im Besitz dieses Schlüssels ist (andernfalls wäre der Antragsteller nicht in der Lage gewesen, diese Signatur zu erstellen).

An dem Datum, das dem Attribut "notAfter" entspricht, wird das Zertifikat als "abgelaufen" betrachtet. Der Ablauf ist ein Mechanismus, der auch dazu dient, sicherzustellen, dass dieselben asymmetrischen Schlüssel, die mit dem digitalen Zertifikat verbunden sind, nicht über einen zu langen Zeitraum verwendet werden können, was sie einer Reihe von kryptoanalytischen Angriffen aussetzen könnte.

Der Verfall kann sich beziehen auf:

- **das Zertifikat** (der Assoziations-Subjekt-Schlüssel)

- **die zugehörigen Schlüssel** in Bezug auf ihren Verwendungszweck (ein Schlüssel kann nur für bestimmte Ziele verwendet werden und für andere nicht)

Ein abgelaufenes Zertifikat kann von seinem Inhaber nicht mehr verwendet werden, da es abgelaufen ist, ist seine Verwendung in der Praxis verboten. In Wirklichkeit hindert den Inhaber des Zertifikats nichts daran, es für andere Zwecke zu verwenden, aber alle, die mit ihm kommunizieren, werden irgendwie feststellen, dass dieses Zertifikat abgelaufen ist und es nicht akzeptieren. Das Zertifikat steht natürlich zur Überprüfung und Entschlüsselung von Nachrichten zur Verfügung, die vor Ablauf der Gültigkeit ausgestellt wurden. In einigen Fällen ist es möglich, ein Zertifikat zu "erneuern", was in der Regel nicht automatisch geschieht.

Manchmal kann ein Zertifikat vor dem Ablauf durch *Widerruf* ungültig gemacht werden (dies geschieht vor dem Wert des Attributs "notAfter"). Der Grund dafür kann z. B. sein, dass der mit dem Zertifikat verbundene Schlüssel kompromittiert wurde, dass das Subjekt nicht mehr mit dem Unternehmen verbunden ist oder die Einheit geändert wurde, dass die Zertifizierungsstelle, die das Zertifikat ausgestellt hat (oder eine ihrer übergeordneten Zertifizierungsstellen), kompromittiert wurde oder andere Gründe. Ein widerrufenes Zertifikat kann nicht mehr verwendet werden, es sei denn, es wird nur "in der Warteschleife" gehalten. Dabei handelt es sich um eine besondere Form des Widerrufs, bei der das Zertifikat aus bestimmten Gründen (z. B. "mögliche Schlüsselkompromittierung", "Antrag auf Widerruf durch eine andere Person als den Eigentümer", "vorübergehende Nichtverwendung des Zertifikats", z. B. wegen Urlaubs) usw. vorübergehend in die CRL aufgenommen wird und seine Gültigkeit später wiederhergestellt werden kann. Im Falle einer "Reaktivierung" aus der Sperrung wird ein neuer Eintrag in die CRL mit einem speziellen Grundcode eingefügt: *removeFromCRL*, da kein Eintrag jemals aus der CRL gelöscht werden kann.

Um die PKI-Benutzer darüber zu informieren, dass ein Zertifikat seine Gültigkeit verloren hat, wird seine Seriennummer in einer *Zertifikatswiderrufsliste* (CRL) aufgeführt. Diese Liste wird in regelmäßigen Abständen von der ausstellenden Zertifizierungsstelle herausgegeben und (aktiv oder passiv) an alle Benutzer weitergegeben, so dass alle Personen, die ein solches Zertifikat besitzen, es nicht mehr verwenden können. Die CRL entspricht dem ITU-T X.509 Standard - rfc3280 - rfc4325, was bedeutet, dass ihr Format standardisiert ist. Sie wird in regelmäßigen Abständen von der Zertifizierungsstelle signiert und enthält ein Feld mit der Bezeichnung "nextUpdate", das das Datum und die Uhrzeit enthält, zu der die neue Zertifikatswiderrufsliste verfügbar sein wird. Wenn dieses Datum in der Vergangenheit liegt, bedeutet dies, dass unsere CRL "abgelaufen" ist, wenn das Datum in der Zukunft liegt, wissen wir, wann wir unsere Liste der widerrufenen Zertifikate aktualisieren müssen. Jedes gesperrte Zertifikat enthält den Grund für die Sperrung. Die CRL wird als Dienst über das LDAP-Protokoll bereitgestellt.

Da CRLs recht umfangreich werden können, insbesondere wenn die Zertifizierungsstelle Hunderttausende oder sogar Millionen von Nutzern hat, kann sie aufgeteilt werden, und die

Zertifizierungsstelle stellt nur Teile der Zertifikatswiderrufsliste zur Verfügung, die sich beispielsweise auf einen bestimmten Zeitraum beziehen. Wenn es also notwendig ist, ein Zertifikat zu überprüfen, lädt der Bürger nur den Teil der CRL herunter, der ihn interessiert. Anstatt eine CRL oder einen Teil davon herunterzuladen, kann ein Dienst, der das *Online Certificate Status Protocol* (OCSP - RFC 2560) implementiert, verwendet werden, um festzustellen, ob eine bestimmte Zertifikatsnummer in der CRL enthalten ist. Die Antworten werden von der Zertifizierungsstelle signiert, damit sie vertrauenswürdig sind.

Die Kette des Vertrauens

Um ein Zertifikat zu überprüfen, muss ein Subjekt die gesamte Kette der zwischengeschalteten CAs überprüfen, ausgehend von einer vertrauenswürdigen CA, möglicherweise der Stamm-CA. Zunächst wird das zu prüfende Zertifikat mit der ausstellenden CA abgeglichen, dann wird das CA-Zertifikat mit der übergeordneten CA abgeglichen, und so weiter bis zu einer (vom Benutzer) vertrauenswürdigen Zwischen-CA oder der Root-CA selbst.

KeyUsage

Schlüssel können für verschiedene Zwecke verwendet werden, und ein Zertifikat legt die zulässigen Verwendungszwecke fest, darunter auch die folgenden:

- **Unverfälschbarkeit**: der Schlüssel kann zur Erstellung einer rechtsgültigen Unterschrift verwendet werden, die dann Dritten gegenüber als handschriftliche Unterschrift verwendet werden kann
- **dataEncipherment:** der Schlüssel kann zur Verschlüsselung von Daten verwendet werden
- **keyAgreement:** der Schlüssel kann für den sicheren Austausch von Schlüsseln verwendet werden

Zeitstempel ing

Eine PKI kann auch dazu verwendet werden, ein Dokument mit einem Zeitstempel gemäß dem Time Stamp Protocol (RFC 3161) zu versehen, um seine Existenz zu einem bestimmten Zeitpunkt in der Vergangenheit zu beweisen. Eine CA signiert das Dokument und versieht es mit einem Zeitstempel. Die Signatur stellt auch sicher, dass das Dokument nach dem Anbringen des Zeitstempels nicht mehr verändert wurde. Ein Zeitstempel kann die Gültigkeit einer elektronischen Signatur über die Lebensdauer des digitalen Zertifikats, das mit den verwendeten Schlüsseln verbunden ist, hinaus verlängern, indem z. B. für jedes Jahr, das vergeht, ein neuer Zeitstempel auf dem Vertrag angebracht wird. Die Anbringung erfolgt jedes Jahr mit kryptografischen Techniken, deren Stärke für den Zeitraum ausreicht, in dem die Signatur selbst angewendet wird.

Standardmechanismen und -formate

Im Folgenden werden die beiden gängigen Standardmechanismen und -formate beschrieben.

PKCS #7 - Kryptografiestandard für öffentliche Schlüssel #7

PKCS #7 beschreibt das Format von Nachrichten/Dateien, die signiert, verschlüsselt (mit symmetrischer und asymmetrischer Verschlüsselung) und durch MAC (digested data) verifizierbar sind.

S/MIME - Sichere Mehrzweck-Mail-Erweiterung

Secure Multipurpose Mail Extension (RFC 3851 - SMIMEv3) ist eine Erweiterung des MIME-Standards (RFC 2045, 2046 und 2047 als Erweiterung von RFC 822) und beschreibt die Verschlüsselung von E-Mail-Nachrichten. PKCS #7-Entitäten werden im Wesentlichen in neue MIME-Entitäten übersetzt und beschreiben daher neue Container, in denen sich Teile von signierten oder verschlüsselten E-Mail-Nachrichten oder signierten oder verschlüsselten Anhängen befinden können.

Austausch- und Speicherformate

Der **PKCS #12 (Personal Information Exchange Syntax Standard #12)** definiert ein Format für den Austausch und die sichere Speicherung (in Dateien) von kryptografischen Schlüsseln und digitalen Zertifikaten (vertrauenswürdige Zertifikatskette). Die Speicherung ist sicher, weil sie auf der Verschlüsselung sowohl der Schlüssel als auch der Zertifikate und der gesamten Kette von Zertifikaten beruht, die ein Subjekt als vertrauenswürdig einstuft. Es ist das Format, das von Browsern, E-Mail-Clients und Betriebssystemen verwendet wird, wenn ein Benutzer eine Signatur und einen Verschlüsselungsschlüssel sowie ein Zertifikat exportiert, beispielsweise für eine Sicherungskopie oder um es auf ein anderes Gerät zu kopieren.

Ein PKCS #12 verschlüsselt die Daten mit einem symmetrischen Algorithmus, dessen Schlüssel aus einem vom Benutzer angegebenen Passwort abgeleitet wird. Es garantiert die Integrität des Inhalts durch HMAC oder durch eine digitale Signatur.

Die Vorteile der Verwendung von PKCS #12 sind:

- es fallen **keine Implementierungskosten an**, da es sich um ein Format handelt, das in vielen Bibliotheken für viele Geräte (vom Computer bis zum Mobiltelefon) verfügbar ist
- sie werden **bereits von den meisten Betriebssystemen** und Softwareprogrammen **unterstützt**, d.h. sie sind sofort verfügbar
- es **werden "starke" symmetrische Algorithmen verwendet** (die angewandte Verschlüsselung ist eine starke, sehr widerstandsfähige Verschlüsselung, d. h. es handelt sich um Schlüssel, die z. B. Brute-Force-Angriffen widerstehen
- sie ist **auf verschiedene Geräte übertragbar**, es ist möglich, eine Datei auf andere Geräte zu verschieben oder sie auf alle von uns verwendeten Geräte zu kopieren, um die Schlüssel immer verfügbar zu haben

Es gibt auch einige Nachteile:

- **Sie können gelöscht, kopiert und übertragen werden**. Dies ist ein wichtiges Problem, da sie gestohlen, gelöscht, kopiert oder einfach unzugänglich gemacht werden können.
- **der private Schlüssel muss direkt von der Signier-/Entschlüsselungssoftware verwendet werden**, d.h. die Software entschlüsselt diese Datei und greift direkt auf den Schlüssel zu, so dass es einen Moment gibt, in dem der Schlüssel ungeschützt auf dem System liegt, obwohl es sich vielleicht nur um den flüchtigen Speicher handelt, der der am wenigsten leicht zugängliche Bereich ist; er ist jedoch in irgendeiner Weise im Klartext auf dem System verfügbar, und dies ist eine sehr starke Einschränkung dieses Formats
- **es ist möglich, Brute-Force-Angriffe durchzuführen**, z. B. wenn jemand die PKCS #12-Datei stiehlt, kann er alle möglichen Passwörter in einem anderen Computersystem ausprobieren

Kryptographische Token

Kryptografische Token sind Hardware-Geräte, die Verschlüsselungsalgorithmen von Haus aus implementieren. Sie erlauben keinen direkten Zugriff auf kryptografisches Material von außen, sondern sind die einzige Möglichkeit, auf den Schlüssel zuzugreifen und ihn zu verwenden. Außerdem können diese Befehle in der Regel nur ausgeführt werden, nachdem sich der Benutzer oder das System im Namen des Benutzers gegenüber dem Token authentifiziert hat, beispielsweise durch Eingabe einer PIN. Token können über ein kleines Betriebssystem verfügen, über das der Benutzer die Schlüsselerzeugung, die Verschlüsselung und die Datensignaturoperationen anfordert. Die sichere Zerstörung des Tokens bedeutet oft nicht nur die Löschung, sondern auch die Unmöglichkeit, auf die Speicherbereiche zuzugreifen, in die der Schlüssel geschrieben wurde, um so jede Form von Angriff zu vermeiden, die wir uns heute nicht vorstellen können. Das Betriebssystem und die Token-Hardware können nach bestimmten Sicherheitsstandards zertifiziert werden.

Der Challenge-Response-Mechanismus

Ein Challenge/Response-Mechanismus ermöglicht es zwei Parteien, sich zu authentifizieren, ohne nützliche Informationen über den Kommunikationskanal preiszugeben. Die Idee ist, dass eine Partei A die andere Partei B "herausfordert", eine Zufallszahl (*Nonce* genannt, weil es sich um eine Zahl handelt, die nur einmal verwendet wird) mit ihrem eigenen Schlüssel zu bearbeiten, z. B. mit einer Einwegfunktion H. Dann gibt B H(nonce) an A zurück. Wenn B das zurückgibt, was A erwartet, ist sich A der Identität von B sicher. Im einfachsten Fall verschlüsselt B die Nonce mit einem geheimen Schlüssel, der A bekannt ist. A, der über denselben Schlüssel verfügt, wendet seinerseits die Funktion mit demselben Schlüssel auf die Nonce an und prüft, ob die ihm von B gegebene Zahl mit der von ihm berechneten übereinstimmt.

Dabei handelt es sich um eine Authentifizierung auf der Grundlage eines Passworts, das nicht über den Kanal gesendet wurde, so dass es nicht möglich ist, den Schlüssel zurückzuverfolgen. Der Mechanismus ist deshalb so wirksam, weil die Zufallszahl bei jeder Sitzung anders ist. Da die über den Kanal ausgelesenen Informationen bei jeder Sitzung anders sind, sind sie für eine spätere Authentifizierung nicht nützlich. Ein Vorteil ist, dass keine Synchronisierung erforderlich ist (anders als beim One-Time-Password-Mechanismus, bei dem es wichtig ist, den Startpunkt des zu verwendenden Pads zu kennen). Da es von Menschen nicht verwendet werden kann, wird es in der Regel für die Authentifizierung zwischen Maschinen verwendet. Ein Challenge-Response-Mechanismus findet sich in den Netzauthentifizierungsmechanismen von ADSL-Routern gegenüber dem Authentifizierungsserver des Anbieters.

Fragen:

In einer separaten Datei.

Anhänge:

Powerpoint in einer separaten Datei.

Referenzen:

Für jedes Thema gibt es eine eigene Seite auf Wikipedia, die für die Zwecke dieses Kurses manchmal schon zu ausführlich ist; Bücher sind zu spezialisiert und nur nach einer sehr tiefen Einarbeitung in das jeweilige Thema nützlich.