11

Asymmetrische Kryptografie

11.1 Nachteile symmetrischer Verfahren

Schlüsseltausch

Obwohl symmetrische Methoden in Netzwerken inzwischen bewährt und weit verbreitet sind, haben sie alle einen gravierenden Nachteil: Sender und Empfänger müssen über den gemeinsamen geheimen Schlüssel verfügen, mithilfe dessen die Nachricht ver- bzw. entschlüsselt werden kann.

Ist Alice räumlich von Bob getrennt und besteht nun der Bedarf, eine Nachricht zum Schutz vor unbefugtem Mitlesen verschlüsselt zu übertragen, so stellt sich die Frage: Wie kann Alice den Schlüssel selbst sicher übertragen?

Ein Problem der symmetrischen Algorithmen ist, dass der Schlüssel selbst für seinen Transport einen sicheren Kanal benötigt. Wenn Sie annehmen, dass Sie Verschlüsselung benutzen müssen, weil Sie wichtige Nachrichten über einen unsicheren Kanal schicken müssen und befürchten, dass diese Nachrichten kompromittiert werden könnten, stellt sich die Frage, über welchen sicheren Kanal Sie den benötigten gemeinsamen Schlüssel übertragen können.

Hätten Sie diesen sicheren Kanal für den Austausch des Schlüssels, so könnten Sie auch auf die Idee kommen, die Nachricht selbst über die sichere Verbindung zu schicken – damit würden Sie sich den Aufwand für die Verschlüsselung sparen, und die Nachricht wäre trotzdem sicher beim Empfänger.

In der Regel werden symmetrische Verfahren dennoch angewandt, weil die Schlüssel noch einen Vorteil gegenüber der Nachricht haben: Sie sind in den meisten Fällen deutlich kürzer als die Nachricht, wenn Sie einmal von Chiffren wie der Vernam-Chiffre absehen. Einen sicheren Kanal für einen kurzen Schlüssel zu finden, ist in der Regel einfacher, als einen sicheren Kanal für eine lange Nachricht.

Trotzdem stellt der sichere Austausch des Kommunikationsschlüssels mitunter ein Problem dar. Was ist, wenn sich Alice und Bob, als sie noch zusammen im selben Raum waren, nicht auf einen geheimen Schlüssel geeinigt haben? Oder Alice und Bob kennen sich im realen Leben gar nicht, sondern haben vor, jetzt das erste Mal über das Internet elektronisch in Kontakt zu treten.
Wie sollen Sie in einem Online-Shop mithilfe einer verschlüsselten Verbindung sicher einkaufen können, wenn Sie noch nie vorher mit diesem Shop einen Schlüssel vereinbart haben? Die Beantwortung dieser Fragen hat einen wichtigen Einfluss auf viele Teilbereiche der Netzwerkkommunikation.

Schlüsselmanagement

Nehmen Sie an, Alice, Bob und Carol wollen in Zukunft sicher miteinander kommunizieren. Sie möchten, dass jeder mit jedem eine verschlüsselte Verbindung aufbauen kann. Die Schlüssel sollen aber jeweils nur zwischen zwei Partnern gültig sein.

Alice vereinbart also mit Bob einen Schlüssel 1 und mit Carol einen Schlüssel 2. Damit Bob kommunizieren kann, benutzt er für Alice Schlüssel 1 und für Carol einen neuen Schlüssel 3. Carol wiederum benutzt für Alice den Schlüssel 2 und für eine Verbindung mit Bob den Schlüssel 3.

Jeder besitzt nun 2 Schlüssel und maximal 2 Personen kennen denselben Schlüssel:

- Alice 1, 2
- Bob 1, 3
- Carol 2, 3

Bei drei Personen existieren also 3 Schlüssel, jeder der Beteiligten muss sich 2 Schlüssel merken. Lassen Sie nun 10 Teilnehmer vereinbaren, ihre Kommunikation gegenseitig mit Schlüsseln zu sichern. Jeder der 10 Teilnehmer benötigt 9 Schlüssel für seine möglichen Partner. Das sind 10 * 9 Schlüssel. Da jeweils zwei Teilnehmer einen Schlüssel gemeinsam benutzen, ist die Anzahl der existierenden Schlüssel 45. Befinden sich 100 Teilnehmer in diesem Netzwerk, so ist die Anzahl der existierenden Schlüssel 4950. Jeder einzelne Teilnehmer muss sich 99 Schlüssel merken.

Die Anzahl der Schlüssel wächst quadratisch, da für n Teilnehmer n*(n-1)/2 Schlüssel benötigt werden. Spätestens hier wird klar, warum ein System mit vorher ausgehandelten symmetrischen Schlüsseln (pre-shared keys) nur bei kleinen Teilnehmerzahlen sinnvoll ist. Für das Internet mit seiner unüberschaubaren Anzahl an Teilnehmern wäre so etwas undurchführbar.

11.2 Einwegfunktion

Die Anfänge der Public-Key-Verschlüsselung

1976 schlugen **Whitfield Diffie** und **Martin E. Hellman** in ihrer Arbeit „New Directions in Cryptography“ einen neuen Ansatz vor, in dem sie das Konzept der Public-Key-Cryptography entwarfen.

Ein Teilnehmer in einem **Public-Key**-Verfahren besitzt einen **public** (öffentlichen) und einen **private** (privaten) Schlüssel. Der öffentliche Schlüssel wird zum Verschlüsseln der an ihn gerichteten Nachrichten durch den Absender verwendet. Der geheime private dient zum Entschlüsseln dieser Nachrichten durch den Empfänger. Während der öffentliche Schlüssel in einem jedermann zugänglichen Verzeichnis öffentlich gemacht wird, muss der private Schlüssel geheim gehalten werden. Dies ist das Grundkonzept von **asymmetrischen Verfahren**.

Damit Public-Key-Verfahren funktionieren können, müssen bestimmte Voraussetzungen gegeben sein. Vor allem taucht hier der Begriff **Einwegfunktion** als zentrales Thema auf.

Eine Einwegfunktion f(x) ist eine Funktion, die eine gegebene Eingabe x auf den Wert y so abbildet, dass es

- ein effizientes Verfahren gibt, die Werte y = f(x) für alle gegebenen x zu berechnen;
- kein effizientes Umkehrverfahren gibt, für alle gegebenen y, die durch f(x) berechnet wurden, das x zu bestimmen.

Es soll also bei einer Einwegfunktion nicht möglich bzw. nicht effizient möglich sein, die Umkehrfunktion zu berechnen. Die Zerlegung von Zahlen in ihre Primfaktoren ist ein (in der Kryptografie nicht ohne Grund) oft zitiertes Beispiel. Große Primzahlen zu multiplizieren und das Ergebnis festzustellen, ist relativ schnell möglich. Dahingegen beansprucht eine Zerlegung einer großen Zahl in ihre Primfaktoren mitunter sehr viel Rechenzeit. Es ist bislang kein effizienter Algorithmus hierfür bekannt.

Ein Problem ist allerdings, dass nicht bewiesen ist, ob es nicht doch eine sehr viel effizientere Methode zur Faktorisierung gibt als die bisher bekannten. Neue Erkenntnisse auf dem Gebiet der Quantencomputer könnten also die etablierten Systeme durchaus in Gefahr bringen.

Echte Einwegfunktionen machen für die Übertragung von Nachrichten keinen Sinn. Würde eine Nachricht X mit einer derartigen Funktion verschlüsselt, sodass der Empfänger Y erhält, so hat dieser ja per Definition keine Möglichkeit, aus dem Y wieder das ursprüngliche X herzuleiten.

Für die Kryptografie werden hier sogenannte Trapdoor-Einwegfunktionen benötigt (Trapdoor, hier: Geheimgang, Hintertür). Diese haben folgende Eigenschaften:

- Es gibt effiziente Verfahren, um y= f(x) zu berechnen.
- Die Berechnung der Umkehrfunktion f^{-1} und somit $x = f^{-1}(y)$ ist nur effizient mithilfe einer geheimen Zusatzinformation möglich (die Hintertür).

So schwer es ist, endgültig zu beweisen, dass es effizientere Methoden zur Faktorisierung gibt, so schwer ist es, zu beweisen, dass es Trapdoor-Einwegfunktionen gibt. Diffie und Hellman haben in ihrer Arbeit ausführlich erörtert, dass die Public-Key-Kryptografie funktionieren würde, wenn es Trapdoor-Einwegfunktionen gäbe. Trotz intensiver Suche konnten sie jedoch keine finden.

Sie schlugen aber ein kryptografisches Protokoll vor, das nach ihnen benannt heute zu den wichtigsten Protokollen der Kryptografie gehört: den **Diffie-Hellman**-Schlüsseltausch.

Restwertoperation, Modulo

Für das Verständnis der folgenden Abschnitte sollten Sie wissen, was eine Modulo-Operation ist. Das Ergebnis einer mathematischen Operation modulo einer Zahl n ist gleich dem ganzzahligen Restwert des Ergebnisses einer Division durch n. Bei symmetrischen Algorithmen kommt oft ein Spezialfall der Modulo-Operation zur Anwendung: Hier wird nur bitweise Modulo 2 addiert, was der Operation XOR entspricht.

Allgemeiner definiert gilt für Modulo-Operationen: Während 11 + 7 = 18 ist, wäre 11 + 7 mod 8 = 2. Die Zahl 18 ist 2x ganzzahlig durch 8 teilbar – der Rest ist 2 und somit das Ergebnis dieser Modulo-Operation. Die Zahl, die die Modulo-Operation definiert (hier 8), wird **Modul** genannt.

5 * 3 = 15; 5 * 3 mod 9 = 6; 15 kann nur 1x durch 9 ganzzahlig geteilt werden. Der Rest ist 6.

1 + 1 = 2; 1 + 1 mod 2 = 0. Das Ergebnis 2 ist ganzzahlig durch 2 teilbar – der Rest ist 0.
Dies entspricht der XOR-Operation.

Wenn Sie also irgendeine Zahl x mod n berechnen wollen, teilen Sie x durch n, und der ganzzahlige Rest ist das Ergebnis.

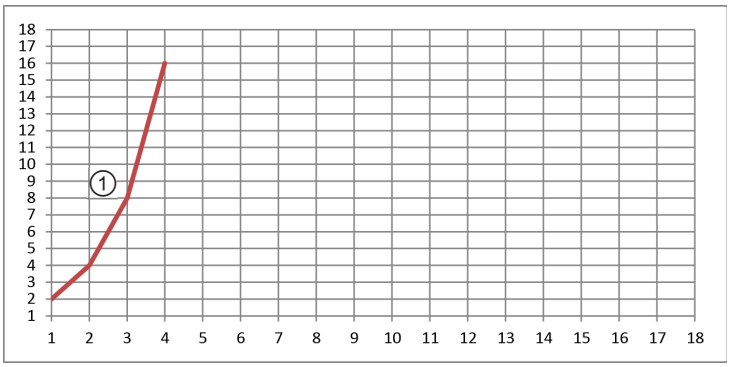
Diskrete Exponentialfunktion und diskreter Logarithmus

Eine gewöhnliche Exponentialfunktion a^b wird durch eine angefügte Modulo-Operation zu einer diskreten Exponentialfunktion. Diese weist in Bezug auf die Suche nach einer Einwegfunktion interessante Eigenschaften auf.

Für die Exponentialfunktion 2^x können Sie folgende Wertetabelle aufstellen, wenn Sie x ganzzahlig von 1 bis 18 steigen lassen:

x	1	2	3	4	5	6	7	8	9
y	**2**	4	8	**16**	32	64	**128**	256	512
x	10	11	12	13	14	15	16	17	18
y	**1024**	2048	4096	**8192**	16384	32768	**65536**	131072	262144

In einer Grafik lassen sich diese Ergebnisse so veranschaulichen ①. Die Funktion steigt sehr schnell und steil an, und ihr Verlauf ist stetig.



Exponentialfunktion 2^x

Wenn Sie diese Funktion umkehren wollen, d. h. aus den gegebenen y-Werten die ursprünglichen x-Werte errechnen wollen, so können Sie schon in der Grafik absehen, wie die Funktion aussehen wird: Es genügt eine Spiegelung an der 45°-Achse durch den Nullpunkt.

Auch mathematisch gibt es bei dieser Funktion keine Probleme, eine Umkehrfunktion zu ermitteln – den Logarithmus. Im Falle von $y = 2^x$ ist die Umkehrfunktion also $x = \log_2 y$.

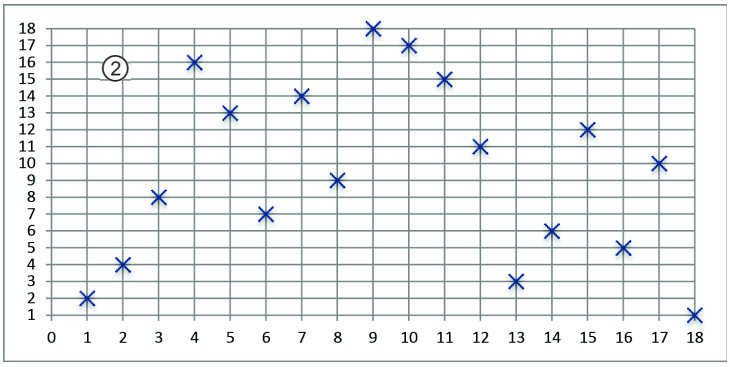
Falls Sie dies auf Ihrem Taschenrechner ausprobieren möchten, denken Sie daran, dass Sie den Logarithmus zur selben Basis verwenden müssen wie die Basis der Exponentialfunktion. Für dieses Beispiel benötigen Sie also den Logarithmus zur Basis 2.

Da Sie auf Kalkulatoren mitunter den natürlichen Logarithmus (ln) und den Logarithmus zur Basis 10 (log) finden, müssen Sie Ihre Ergebnisse umrechnen: Sie wollen zu einer Zahl x den Logarithmus zur Basis b errechnen, können aber nur den Logarithmus mit Basis a anwenden: $\log_b x = \log_a x / \log_a b$. So können Sie relativ leicht errechnen, dass bei gegebenem y von 32768 folgt: log 32768 / log 2 = 15. Das ursprüngliche x war also gleich 15.

Wenn Sie die diskrete Exponentialfunktion 2^x mod 19 bilden, so erhalten Sie folgende Wertetabelle:

x	1	2	3	4	5	6	7	8	9
y	**2**	4	8	**16**	13	7	**14**	9	18
x	10	11	12	13	14	15	16	17	18
y	**17**	15	11	**3**	6	12	**5**	10	1

In einer Grafik dargestellt, zeigt sich folgende Verteilung ②. Abgesehen von den ersten vier Punkten ist für das menschliche Auge kein regelmäßiges Verteilungsschema zu erkennen.



Diskrete Exponentialfunktion 2^x mod 19

Die Umkehrung mithilfe eines Rechners zu ermitteln, würde im naiven Fall für den Programmierer heißen, bei gegebenem y schrittweise alle diskreten Exponentialfunktionen zu 2^x durchzurechnen und zu prüfen, ob das Ergebnis der Berechnung gleich y ist – dann wurde die Umkehrung gefunden. Obwohl es Verfahren gibt, mit denen die Suche nach einem diskreten Logarithmus beschleunigt wird, ist diese immer noch um ein Vielfaches aufwendiger als die Bildung der entsprechenden Exponentialfunktion durch eine Reihe simpler Multiplikationen.

In diesem Sinn ist die diskrete Exponentialfunktion also eine Einwegfunktion.

Erzeugende Elemente

Ein erzeugendes Element der Addition ist die 1. Betrachten Sie den Zahlenraum Z mit Elementen von 0 bis 9 bei der Addition modulo 10, so gilt:

1 = 1, 1 +1 = 2, 1 + 1 + 1 = 3
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 9
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 0

Sie können also durch wiederholte Addition von 1 alle Elemente in der Gruppe erzeugen. Deswegen wird die 1 ein **erzeugendes Element** der additiven Gruppe genannt. Betrachten Sie stattdessen die sogenannte multiplikative Gruppe Z, die durch Multiplikation modulo n entsteht (das entspricht der diskreten Exponentialfunktion), so existieren auch hier erzeugende Elemente. Obwohl hier die Verhältnisse deutlich komplizierter sind, lässt sich mathematisch nachweisen, wann ein Element a ein erzeugendes Element einer multiplikativen Gruppe modulo n ist. Voraussetzung dafür ist, dass Modul n eine Primzahl ist.

Betrachten Sie noch einmal das Beispiel oben. Sie werden feststellen, dass die 2 ein erzeugendes Element der Gruppe Z_{19} ist: Als Ergebnisse der Berechnung 2^x mod 19 erhalten Sie, wenn Sie alle x von 1 bis 18 durchlaufen haben, sämtliche Zahlen von 1 bis 18 – jeweils **genau ein Mal**. Diese Eigenschaft hat äußerst interessante Nebenwirkungen für die Kryptografie. In gewisser Weise ordnet die Tabelle von 2^x mod 19 den x-Werten von 1 bis 18 neue y Werte (ebenfalls von 1 bis 18) in anderer Reihenfolge zu. Dies kann auch als Permutationstabelle verwendet werden.

Ein Alphabet, das Sie mit 2^x mod 19 permutieren wollen, dürfte maximal 18 Zeichen haben, da die Gruppe nicht mehr Elemente besitzt. Dieses Problem können Sie jedoch dadurch lösen, dass Sie eine Gruppe mit mehr Elementen finden.

11.3 Diffie-Hellman-Schlüsseltausch

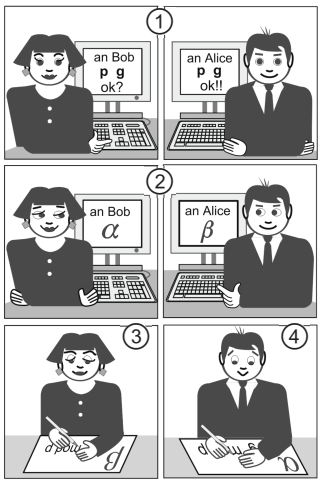
Ein Protokoll zum geheimen Schlüsseltausch

Diffie und Hellman waren zwar bei ihrer Suche nach einer Trapdoor-Einwegfunktion nicht erfolgreich. Gleichzeitig wurden sie aber auf die interessanten Eigenschaften der diskreten Exponentialfunktion aufmerksam und entwickelten ein Protokoll, das die Eigenschaften der diskreten Exponentialfunktion nutzt, um zwischen zwei Partnern sicher einen gemeinsamen Schlüssel zu ermitteln.

Aus der symmetrischen Kryptografie wissen Sie, dass ein gemeinsamer geheimer Schlüssel für das Ver- und Entschlüsseln erforderlich ist. Allerdings stellt sich das Problem, dass meist kein sicherer Kanal für den Schlüsseltausch vorhanden ist.

Ablauf des Schlüsseltauschs

Diffie und Hellman schlagen folgendes Protokoll vor:



Alice und Bob einigen sich beide auf eine Primzahl p und eine Zahl g ①. Die Zahl g muss erzeugendes Element der Gruppe Z_p sein. Nun wählt Alice für sich eine Zahl a und Bob eine Zahl b. Diese beiden Zahlen halten Alice und Bob jedoch geheim.

Alice berechnet: $\alpha = g^a \bmod p$.
Bob berechnet: $\beta = g^b \bmod p$.

Beide tauschen nun die soeben errechneten α und β ②.

Alice potenziert nun das von Bob erhaltene β mit ihrer geheimen Zahl a ③.

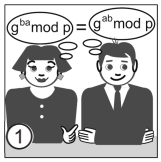
Bob potenziert das α von Alice mit seiner geheimen Zahl b ④.

Was haben Alice und Bob gerade berechnet?

Alice berechnet: $\beta^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p$.
Bob berechnet: $\alpha^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$.

Da für die beiden Exponenten das Kommutativgesetz gilt, ist $g^{ba} \bmod p = g^{ab} \bmod p$.

Diese gemeinsam ermittelte Zahl ist nun der neue Schlüssel für die weitere Kommunikation.



Alice und Bob haben also beide gerade dieselbe Zahl berechnet ①. Das Ergebnis dieser Berechnungen wurde niemals über einen öffentlichen Kanal übertragen und ist somit nur Alice und Bob bekannt. Wollte Eve den Schlüssel ebenfalls berechnen, so müsste sie diesen aus den öffentlich übertragenen Größen g, p, α und β selbst berechnen ②. Diese Berechnung allerdings läuft darauf hinaus, dass Eve entweder den diskreten Logarithmus von α oder aber β berechnen muss, um entweder a oder b zu erfahren. Wie Sie bereits gesehen haben, ist dies mitunter sehr schwierig.

Die Sicherheit des DH-Schlüsseltausches hängt davon ab, in welcher Größenordnung die von Alice und Bob gewählte Zahl g und die Primzahl p liegen. Je größer diese beiden Zahlen gewählt werden, desto schwieriger wird es für Eve, den gemeinsamen Schlüssel zu errechnen.

Anwendungsmöglichkeiten des DH-Schlüsseltauschs

Alice und Bob können den gemeinsamen Schlüssel, der ja eine natürliche Zahl ist, als Schlüssel für ein symmetrisches Verschlüsselungsverfahren ihrer Wahl benutzen. Mithilfe des DH-Protokolls können beliebige Kommunikationspartner, ohne sich vorher getroffen zu haben, über einen unsicheren Kanal einen gemeinsamen Schlüssel aushandeln, wenn sie einen sicheren Kanal benötigen.

11.4 El-Gamal

Ein Public-Key-Verschlüsselungsverfahren

Basierend auf dem Diffie-Hellman-Protokoll schlug Taher El-Gamal eine Methode vor, wie das Diffie-Hellman-Protokoll nicht nur für den sicheren Schlüsseltausch, sondern auch als asymmetrisches Kryptosystem benutzt werden kann. Dabei steht im Mittelpunkt, dass das Generieren und Tauschen von Schlüsseln zeitlich nicht mit der Übermittlung einer Nachricht zusammenfallen muss. Das El-Gamal-Verschlüsselungsverfahren ist auch unter dem Namen **DSA** (Digital Signature Algorithmus) geläufig.

Ablauf von El-Gamal

Sämtliche Teilnehmer dieses Systems müssen sich zuerst auf eine Verschlüsselungsmethode, eine Primzahl p und eine natürliche Zahl g einigen. Anschließend wählt jeder Teilnehmer T eine natürliche Zahl t. Diese betrachtet er als seinen privaten Schlüssel und hält diesen geheim.

Jeder Teilnehmer berechnet mithilfe seiner Zahl t den Wert $\tau = g^t \bmod p$ und stellt diesen Wert in das öffentlich zugängliche Verzeichnis. Aus den öffentlich verfügbaren Daten ist es für Eve nicht möglich, den geheimen Schlüssel t zu errechnen, da sie hierfür den diskreten Logarithmus log(τ) kennen müsste.

Will Alice nun an den Teilnehmer T eine Nachricht senden, so schlägt sie zuerst den öffentlichen Schlüssel τ des Teilnehmers T im Verzeichnis nach ①.

Alice wählt einen zufälligen Wert a ② und berechnet selbst: $\alpha = g^a \bmod p$.

Sie errechnet die Zahl k aus $\tau^a \bmod p$. Diese Zahl (k = key) ist der Schlüssel, den sie für die symmetrische Verschlüsselung ihrer Nachricht benutzt ③.

Alice sendet die nun verschlüsselte Nachricht zusammen mit α an T ④.

Möchte der Empfänger die Nachricht entschlüsseln, so muss er zuerst den Schlüssel k errechnen, den Alice für die Nachricht verwendet hat ⑤. Dazu verwendet er das von Alice an die Nachricht angefügte α.

Er berechnet: $k = \alpha^t \bmod p$ ⑥.

Analog zu Diffie-Hellman gilt auch hier, dass $\alpha^t \bmod p = g^{at} \bmod p$ ist und $\tau^a \bmod p = g^{ta} \bmod p$ ist. Somit haben Alice und der Teilnehmer T denselben Schlüssel k errechnet.

Das System nach El-Gamal benötigt so viele öffentliche Schlüssel, wie es Teilnehmer gibt. Dies erklärt sich dadurch, dass das jeweilige τ des Teilnehmers veröffentlicht wird.

Wählen die Sender der Nachrichten für jede Nachricht eine Zufallszahl (in diesem Beispiel a), so wird jede in diesem System versandte Nachricht mit einem **anderen** Schlüssel verschlüsselt – auch wenn sie an denselben Empfänger geht.

Schwächen von El-Gamal/DSA

El-Gamal/DSA unterstützte über einen langen Zeitraum nur Schlüssel mit einer Länge von nur 1024 Bit. So verwenden nicht mehr aktuelle GnuPG-Implementierungen diese Einstellung. Des Weiteren werden für die Erzeugung **jeder** Signatur ungewöhnliche Zufallszahlen benötigt. Sofern der Generator für die Erzeugung der Zahlen auch nur punktuell kompromittiert wird, ist keine Sicherheit mehr gegeben. Dies kann z. B. durch einen Patch des Quellcodes erzeugt werden. Auch gewöhnliche (in der Normalverteilung öfter genutzte) Zufallszahlen stellen ein Sicherheitsdefizit dar.

11.5 RSA

Forschungsarbeit des RSA-Teams

Dadurch, dass Diffie und Hellman bei ihrer Suche nach einer Trapdoor-Einwegfunktion erfolglos blieben, wurde ein anderes Forscherteam auf diese Thematik aufmerksam: **Ronald Rivest**, **Adi Shamir** und **Len Adleman**. Laut Adi Shamir war das ursprünglich erklärte Ziel, zu beweisen, dass es keine Trapdoor-Einwegfunktionen geben kann.

Ironischerweise sind sie mit diesem Vorhaben nicht nur gescheitert – sie haben sogar bei dem Versuch, die Unmöglichkeit einer Trapdoor-Einwegfunktion nachzuweisen, genau eine solche entdeckt. 1977 entwickelten sie somit das bekannteste Public-Key-Verfahren, das nach den Initialen der drei Erfinder benannt ist: RSA.

Ablauf von RSA

Für RSA benötigen Sie das RSA-Modul, hier als n bezeichnet. Dies ist das Produkt zweier Primzahlen, denn eine große Zahl zu erzeugen, indem man zwei Primzahlen miteinander multipliziert, ist relativ einfach. Die Faktorisierung (Zerlegung einer Zahl in ihre Primfaktoren) des Ergebnisses ist dagegen ungleich aufwendiger.

Neben dem RSA-Modul n benötigen Sie einen Wert e und dessen modulares Invers d. Diese werden wie folgt gebildet:

Es werden zufällig zwei unterschiedliche Primzahlen p und q gewählt. In der Praxis werden hierzu ab einer Zahl der gewünschten Mindestgröße Primzahltests durchgeführt, bis eine Primzahl gefunden wird. Dann wird für eine zweite Mindestzahl, die deutlich größer als die erste Primzahl sein sollte, identisch vorgegangen. Die so erhaltenen Primzahlen werden hier p und q genannt.

z. B. p = 23 und q = 47

Bilden Sie das Produkt n aus den beiden Primzahlen p und q. Diese wird als RSA-Modul bezeichnet. Das RSA-Modul ist ein Teil sowohl des privaten als auch des öffentlichen Schlüssels.

z. B. 23 * 47 = 1081.

Berechnen Sie nun die Euler'sche φ-Funktion (Phi) von n: φ(n) = (p - 1) * (q - 1)

z. B. (23 - 1) * (47 - 1) = 1012

Für den öffentlichen Schlüssel benötigen Sie nun einen Wert e, der teilerfremd zu φ(n) ist. Teilerfremd bedeutet, dass er keinen gemeinsamen Teiler mit φ(n) haben darf (relativ prim). Für diesen muss gelten 1 < e < φ (n). In der Praxis wird hier häufig 2^{16} + 1 = 65537 verwendet.

z. B. e = 3

Berechnen Sie nun für den privaten Schlüssel den Wert d, der das modulare Invers von e darstellt:

(e*d) mod ((p - 1) * (q - 1)) = 1.

z. B. (3 * 675) mod ((23 - 1) * (47 - 1)) = 1

Nun sollten die Primzahlen p und q und die φ(n) sicher gelöscht werden, um die Schlüsselkomponenten n, e (öffentlich) und n, d (privat) zu schützen.

Zum Verschlüsseln wird nun die Nachricht m mit dem öffentlichen Schlüssel e potenziert und anschließend modulo n angewandt: m^e mod n = c. Für die Klartextnachricht "5" wäre die verschlüsselte Nachricht c in unserem Beispiel:

5^3 mod 1081 = 125

Der Empfänger entschlüsselt nun die Nachricht c, indem er die verschlüsselte Nachricht c mit seinem geheimen Schlüssel d potenziert und anschließend modulo n anwendet: c^d mod n = m

125^{675} mod 1081 = 5

Wie Sie sehen, ist das RSA-Verfahren eine echte Trapdoor-Einwegfunktion. Das Potenzieren einer Nachricht modulo n wäre nur durch den diskreten Logarithmus umkehrbar, dessen Berechnung bei großem n extrem viel Zeitaufwand mit sich bringen würde. Das Trapdoor ist hier das Potenzieren mit der geheimen Zusatzinformation, also mit dem geheimen Schlüssel d des Teilnehmers. Als inverses Element zu e gewählt, macht d die Potenzierung der Nachricht mit e rückgängig.

Primzahlen und Sicherheit

Mit obigen Beispielen und Erklärungen wird klar, warum Primzahlen vor allem in der asymmetrischen Kryptografie eine wichtige Rolle spielen. Sie werden einerseits benötigt, um die notwendigen Voraussetzungen für das Erzeugen einer multiplikativen Gruppe und das Vorhandensein eines inversen Elements zu schaffen. Die Schwierigkeit für einen Angreifer, einen RSA-Schlüssel nach dem Abfangen einer verschlüsselten Nachricht zu errechnen, hängt davon ab, ob er die mit dem öffentlichen Schlüssel bekannt gegebene Zahl n (die als Modul benutzt wird) in ihre Primfaktoren p und q zerlegen kann.

Gelingt dies dem Angreifer, so kann er $\varphi(n) = (p - 1) * (q - 1)$ berechnen und anschließend mit der Lösung der Gleichung $e * d = 1 \bmod \varphi(n)$ den geheimen Schlüssel d berechnen. Dann wäre er in der Lage, die für den Teilnehmer verschlüsselten Nachrichten selbst zu entschlüsseln.

Um die Faktorisierung schwierig zu machen, ist es also angebracht, große Primzahlen als Basis für n zu nehmen. Die oft verwendete Bitlänge bei asymmetrischen Algorithmen gibt Auskunft darüber, in welcher Größenordnung die verwendeten Werte sind. Die Bitlängen von symmetrischen und asymmetrischen Schlüsseln sind nicht direkt miteinander zu vergleichen. So kann niemand behaupten, ein RSA-Schlüssel wäre mit 2048 Bit genau 16-mal so sicher wie ein 128-Bit-AES-Schlüssel. Hier spielt die Tatsache eine Rolle, dass von den 2^{128} möglichen AES-Schlüsseln nach dem gegenwärtigen Wissensstand alle als Verschlüsselungsschlüssel infrage kommen. Bei 2^{2048} Möglichkeiten für einen asymmetrischen Schlüssel sind aber nicht alle darstellbaren Zahlen als Schlüssel bzw. Modul verwendbar, da Schlüssel und Module bestimmte Eigenschaften erfüllen müssen.

Rechenzeit und hybride Verschlüsselung

Aufgrund der verwendeten mathematischen Operationen sind asymmetrische Algorithmen bei der Ausführung etwa um Faktor 1000 langsamer als symmetrische Methoden. Letztere arbeiten mit einfachen Bitoperationen, Verschiebungen und Permutationstabellen.

Um die Vorteile beider Verfahren zu nutzen, wird in kryptografischen Systemen gerne auf hybride Lösungen zurückgegriffen. Oft wird ein Public-Key-Verfahren für das Schlüsselmanagement verwendet, zum Versenden der eigentlichen Nachrichten aber ein symmetrisches Verfahren.

El-Gamal kommt diesem Prinzip schon von vorneherein sehr nahe, da in El-Gamal der Diffie-Hellman-Schlüsseltausch benutzt wird, um einen gemeinsamen, geheimen symmetrischen Schlüssel für den Transport der Nachricht zu generieren.

Beim hybriden RSA generiert Alice eine Zufallszahl. Diese Zahl benutzt Alice als symmetrischen Schlüssel (Sitzungs-Schlüssel), um damit eine Nachricht an Bob zu verschlüsseln. Damit Bob diese Nachricht wieder entschlüsseln kann, wird der gerade benutzte Sitzungs-Schlüssel (Session-Key) mithilfe von RSA verschlüsselt und an die Nachricht angehängt. Bob empfängt die Nachricht und kann mithilfe seines privaten Schlüssels den Sitzungs-Schlüssel wieder dechiffrieren. Mit diesem Sitzungsschlüssel entschlüsselt er dann die eigentliche Nachricht.

Der Vorteil dieser Vorgehensweise liegt darin, dass die Nachricht mit einem schnellen symmetrischen Verfahren verschlüsselt werden kann. Das langsame asymmetrische Verfahren wird nur gebraucht, um den vergleichsweise kurzen Sitzungs-Schlüssel zu sichern.

Schwächen von RSA

RSA basiert auf einem Faktorisierungsalgorithmus. Untersuchungen zeigen, dass bei einer Verbesserung der Mechanismen für die Faktorisierung eine Schlüssellänge von 1024 Bit nicht als sicher angesehen werden kann. Dass u. a. die **NSA** (National Security Agency) diese Schlüssel dekodieren kann, darf als möglich angesehen werden, obwohl der explizite Beweis hierzu bisher fehlt. Eine Schlüssellänge von 2048 Bit gilt hier als sicher (Stand 2015). Auch der genutzte Standard **PKCS#1** 1.5 (Public Key Cryptography Standards) ist bei nicht sicherer Implementierung eine Schwachstelle (vgl. Abschnitt 11.8).

11.6 Digitale Signatur

Eine digitale Signatur ist eine Art Unterschrift unter einem digitalen Dokument. Dabei wird mittels kryptografischer Verfahren eine genaue und eindeutige Zuordnung dieses Dokuments zu seinem Absender sichergestellt, ersetzt quasi seine eigenhändige Unterschrift.

Authentifizierung von Nachrichten

Basierend auf dem RSA-Verfahren ist es relativ leicht möglich, eine digitale Signatur zu erstellen:

Alice möchte eine Nachricht digital signieren. Dazu besitzt sie ein Schlüsselpaar d und e, das sie nach RSA erstellt hat.

Alice bildet: $c = m^d \bmod n$. Sie verschlüsselt also ihre Nachricht, allerdings benutzt sie für diese Operation ihren privaten Schlüssel. Die so unterschriebene Nachricht sendet sie an Bob.

Zur Prüfung der Unterschrift benutzt Bob den von Alice erhältlichen oder bereits publizierten öffentlichen Schlüssel. Er berechnet: $m = c^e \bmod n$. Er potenziert die von Alice erhaltene Nachricht mit dem öffentlichen Schlüssel und erhält wieder die Originalnachricht m. Dies ist möglich, da d und e zueinander inverse Elemente sind.

Was hat Bob damit gewonnen? Erhält Bob nach dem Potenzieren der Nachricht mit dem öffentlichen Schlüssel e von Alice die Nachricht, so weiß Bob, dass diese Nachricht nur von Alice signiert worden sein kann. Alice ist die einzige Person, die im Besitz des zu ihrem öffentlichen Schlüssel passenden geheimen Schlüssels ist.

Erhält Bob nach dem Potenzieren keine sinnvolle Nachricht, so wurde nicht der korrekte private Schlüssel benutzt oder die Signatur ist gefälscht worden.

Um die digitale Signatur prüfen zu können, benötigt Bob den öffentlichen Schlüssel von Alice. In diesem Fall ist er sogar gezwungen, die Signatur zu prüfen, da er ohne Entschlüsselung die Nachricht nicht lesen kann. Da es nicht praktikabel ist, die Nachricht in ihrer kompletten Länge so zu signieren, dass ein Lesen ohne Prüfung nicht möglich ist, wäre eine weitere Alternative, die Nachricht einmal im Klartext zu senden und daran die signierte Nachricht anzuhängen. Das würde allerdings ein Anwachsen der Nachrichtenlänge auf das Doppelte bewirken und ist somit auch ungeeignet. Eine Lösungsmöglichkeit bieten sogenannte Hashfunktionen.

11.7 Hashfunktionen

Definition einer Hashfunktion

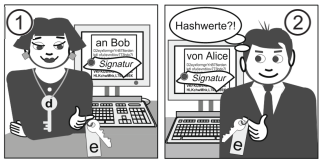
Der Einsatz von sogenannten Hashfunktionen hat sich zur Erkennung von Fehlern im Informatikeinsatz schon länger bewährt und stellt im Zusammenhang mit digitalen Signaturen eine willkommene Alternative dar. Eine Hashfunktion ermittelt aus einer beliebigen Menge an Eingabedaten einen Hashwert. Dieser Hashwert ist quasi eine Prüfsumme und hat immer dieselbe Länge.

Hashfunktionen sind durch die Länge des berechneten Wertes gekennzeichnet. So liefert eine Funktion mit 56 Bit nur 2^{56} verschiedene Hashwerte, während eine Funktion mit 160 Bits theoretisch 2^{160} verschiedene Hashwerte liefern kann.

Sinn eines Hashwertes ist es, als Prüfsumme für eine bestimmte Eingabe (z. B. einen Dateiinhalt oder eine Nachricht) zu fungieren. Wird der Inhalt der Datei oder der entsprechenden Nachricht geändert, so sollte auch der Hashwert ein anderer sein. Die Chance, dass zwei verschiedene Nachrichten existieren, die denselben Hashwert besitzen, sollte möglichst klein sein. Eine größere Anzahl von möglichen Hashwerten wirkt sich hier positiv aus.

Eine Hashfunktion ist in gewisser Weise auch eine Einwegfunktion, da Sie leicht den Hashwert h einer Nachricht m bilden können h = f(m), aber bei gegebenem Hashwert die ursprüngliche Nachricht nicht rekonstruieren können. Eine gute Hashfunktion soll wie ein Fingerabdruck der Nachricht sein – der Fingerabdruck jedes Menschen ist einzigartig, doch aus dem Fingerabdruck selbst kann man den Menschen nicht rekonstruieren.

Für die Erzeugung einer digitalen Signatur kann eine Hashfunktion so eingesetzt werden, dass die Nachricht m unverschlüsselt übertragen wird. Alice bildet den Hashwert h dieser Nachricht und chiffriert diesen mit ihrem privaten Schlüssel d ①.



Bob empfängt nun die Nachricht und kann diese lesen. Möchte er die Unterschrift von Alice prüfen, so entschlüsselt er den angehängten Hashwert mit Alice' öffentlichem Schlüssel. Bob bildet anschließend auch selbst den Hashwert der empfangenen Nachricht ②. Stimmen beide Hashwerte überein, so stammt die empfangene Nachricht wirklich von Alice ③.

Bekannte Hashfunktionen sind u. a.:

- MD5 (1992)
- RIPEMD (1992)
- SHA-1 (1995)
- Tiger (1996)
- SHA-2 (2001)
- SHA-3 (2012)

MD5

Message Digest 5 ist einer der bekanntesten und immer noch genutzten Hashalgorithmen. Ron Rivest, einer der RSA-Erfinder, hat diesen Algorithmus entwickelt. MD5 berechnet 128-Bit-Werte, indem in mehreren Runden, ähnlich wie ein Verschlüsselungsverfahren, die zu hashenden Daten durch eine Reihe binärer Operationen geführt werden.

Obwohl inzwischen von der Verwendung von MD5 aufgrund möglicher Schwachstellen („Geburtstagsangriff“, vgl. Abschnitt 11.8) abgeraten wird, ist MD5 in vorhandenen Systemen aus Gründen der Kompatibilität mitunter nicht zu ersetzen.

SHA

SHA, der Secure Hash Algorithm, wurde 1993 von der NSA entwickelt. Als Grundlage diente der MD4-Algorithmus und Vorläufer von Rivests MD5. Nachdem 1995 ein Sicherheitsproblem gefunden wurde, das nicht veröffentlicht wurde, veröffentlichte die NSA den Standard SHA-1 als Abhilfe, der in dieser Form noch in vielen modernen kryptografischen Paketen zum Einsatz kommt.

SHA-1 berechnet einen 160-Bit-Hashwert. SHA-1 ist derzeit einer der am häufigsten benutzten Hashalgorithmen. Das Ende seiner Nutzungsdauer zeichnet sich jedoch schon ab, da im Februar 2005 durch erfolgreiche kryptografische Angriffe auf diesen Algorithmus die Komplexität deutlich reduziert werden konnte. Mit steigender Rechenleistung der Computer und eventuell weiteren Durchbrüchen in der Kryptologie steigt also die Wahrscheinlichkeit, dass ein Text so gestaltet werden kann, dass eine bestimmte Prüfsumme erzielt wird.

Das amerikanische NIST (National Institute of Standards an Technology) empfiehlt, wenn möglich die Hashfunktion SHA-1 durch Algorithmen der Familie **SHA-2** zu ersetzen, die Prüfsummen mit variabler Länge größer als 160 Bits berechnen. Dementsprechend heißen diese Hashfunktionen dann entsprechend der Hashlänge beispielsweise SHA-256, SHA-384 oder SHA-512.

Der neue Standard **SHA-3** beruhte auf einer Ausschreibung des NIST, die im Jahre 2012 erfolgte. Er nutzt hierbei einen neuen mathematischen Algorithmus, der sich signifikant von SHA-2 unterscheidet. Dabei sind Schlüssellängen von 224, 256, 384 und 512 Bit vorgesehen. Das NIST wollte aus Performanceerwägungen jedoch nur 128 und 256 Bit zum Standard erklären. Massive Widerstände, auch aufgrund der NSA-Aktivitäten, verhinderten dies.

11.8 Schwachstellen in RSA

Wie sicher ist RSA wirklich?

Obwohl Public-Key-Systeme relativ sicher erscheinen, gibt es einige Probleme, die in dieser Form bei symmetrischen Verfahren nicht bekannt waren.

Die Sicherheit asymmetrischer Verfahren beruht auf der Tatsache, dass Alice beim Versenden einer Nachricht an Bob auch sicher sein kann, dass der Schlüssel, den sie zum Chiffrieren an Bob verwendet, auch wirklich Bobs öffentlicher Schlüssel ist.

Haben Alice und Bob sich nicht persönlich getroffen und ihre öffentlichen Schlüssel ausgetauscht, so wird Alice den Schlüssel von Bob aus einem öffentlich zugänglichen Verzeichnis (in der Regel ein Public-Key-Server) beziehen.

Möchte Mallory in Zukunft alle Nachrichten lesen können, die Alice an Bob schreibt, so erzeugt er selbst ein Schlüsselpaar und stellt den öffentlichen Schlüssel unter dem Namen Bob in das Verzeichnis. Wenn Alice nun eine Nachricht schreibt, benutzt sie einen Schlüssel, zu dem Mallory den privaten Schlüssel besitzt und mit dem er somit die Nachrichten entschlüsseln kann.

Mallory könnte sogar noch weiter gehen und Bob ebenfalls seinen eigenen Schlüssel unterschieben. Diesmal würde er diesen Schlüssel als den von Alice deklarieren. In so einem Fall könnte Mallory dann die von Alice kommenden Nachrichten lesen und verschlüsselt an Bob weitersenden. Prüft Bob die Unterschrift der empfangenen Dokumente, so prüft er die von Mallory erzeugte Signatur anhand von Mallorys Public-Key. Bob wäre also in dem fatalen Irrglauben, Alice hätte ihm ein digital signiertes und verschlüsseltes Dokument geschickt.

Dieser Angriff ist auch als „**Man-in-the-Middle-Attack**“ bekannt. Eine gewisse Absicherung gegen die Fälschung von öffentlichen Schlüsseln stellen die verschiedenen Infrastrukturen zur Verteilung von öffentlichen Schlüsseln (Public Key Infrastructure, PKI) dar.

Chosen-Ciphertext-Angriff

Bei einem Chosen-Ciphertext-Angriff möchte Mallory eine Nachricht lesen, die von Alice an Bob geschickt wurde. Dazu kann Mallory das Verschlüsselungssystem kompromittieren, indem er einen Chiffretext seiner Wahl erstellt.

Mallory fängt die verschlüsselte Nachricht c ab, die Alice an Bob mit dessen öffentlichem Schlüssel chiffriert sendet, und besorgt sich den öffentlichen Schlüssel e von Bob. Zusätzlich wählt Mallory eine Zufallszahl r, die kleiner als n und teilerfremd zu n ist. Da er r selbst gewählt hat, kann Bob mithilfe der bekannten Gleichung das inverse Element von r berechnen: r^{-1}.

Zuerst verschlüsselt Mallory das zufällige r:

$x = r^e \bmod n$

Um sein selbst gewähltes r mit der abgefangenen Nachricht verarbeiten zu können, multipliziert er die Nachricht c mit seinem errechneten x:

$y = x * c \bmod n$

Mallory überredet jetzt Bob dazu, die gerade errechnete Nachricht y digital mit seinem privaten Schlüssel zu signieren. Lässt Bob sich darauf ein, so berechnet Bob Folgendes:

$u = y^d \bmod n$, da Bob für eine Signatur mit RSA die Nachricht mit seinem privaten Schlüssel potenzieren muss.

Mallory ist daraufhin in der Lage, mit dem von Bob berechneten Wert weiterzuarbeiten.
Er multipliziert das erhaltene u mit dem inversen Element r^{-1}. Diese Berechnung kann wie folgt aufgelöst werden:

$$\begin{aligned} r^{-1} * u \bmod n &= r^{-1} * y^d \bmod n \\ &= r^{-1} * x^d * c^d \bmod n \\ &= r^{-1} * (r^e)^d * c^d \bmod n \\ &= r^{-1} * r * c^d \bmod n \end{aligned}$$

Da sich die Exponenten e und d gegenseitig genauso aufheben wie auch r und r^{-1}, bleibt schließlich als Einziges auf der rechten Seite Folgendes übrig:

$$= c^d \bmod n$$

Die Chiffrenachricht c potenziert mit dem privaten Schlüssel d von Bob ist aber nichts anderes als die Klartextnachricht. Mallory hat es also geschafft, sich indirekt unter Mitwirkung von Bob einen Text entschlüsseln zu lassen, der nur für Bob bestimmt war. Durch Anwendung seines privaten Schlüssels auf eine ihm unbekannte Nachricht hat Bob Mallory die Arbeit erspart, Modul n in seine Primfaktoren zerlegen zu müssen.

Fazit: Unterschreiben Sie nie irgendwelche binären Daten unbekannten Inhalts mit Ihrem privaten Schlüssel – es könnte sich um einen Chosen-Ciphertext-Angriff handeln.

Der Geburtstagsangriff

Eine weitere Klasse von Angriffen basiert auf dem Phänomen, dass es zwar sehr unwahrscheinlich ist, dass eine bestimmte Person an genau demselben Tag wie Sie Geburtstag hat. Es ist aber durchaus wahrscheinlich, dass in einer Gruppe aus mehreren Personen zwei beliebige Personen am gleichen Tag Geburtstag haben.

Im ersten Fall ist von einer Wahrscheinlichkeit von 1:365 auszugehen. Im zweiten Fall ist die Wahrscheinlichkeit, dass zwei Personen in einem Raum mit 23 Leuten am selben Tag Geburtstag haben, schon ca. 50 %.

Übertragen auf die Kryptografie, Hashwerte und digitale Signaturen bedeutet das: Wenn Sie ein Dokument bzw. dessen Hashwert haben und ein anderes Dokument suchen, das genau denselben Hashwert hat, so ist die Wahrscheinlichkeit, dass Sie ein zweites finden, relativ gering.

Erstellen Sie aber eine große Menge an Dokumenten und suchen nach irgendwelchen zwei Dokumenten, die denselben Hashwert haben, so ist die Wahrscheinlichkeit, dass Sie erfolgreich sind, bedeutend größer. Weisen verschiedene Dokumente den gleichen Hashwert auf, spricht man von einer Kollision.

Dieser Sachverhalt könnte folgendermaßen ausgenutzt werden: Mallory erstellt zwei Versionen eines Dokuments: ein Überweisungsformular, in dem Alice an Bob einen kleinen Betrag überweist, und eine zweite Version, in der Alice an Mallory eine große Summe überweist. Anschließend erzeugt Mallory von jedem Dokument zahlreiche Versionen, die aber denselben Inhalt haben. Variationen könnte Mallory durch das Einfügen bzw. Weglassen von Returns, Leerzeichen und Tabulatoren produzieren, die im endgültigen Dokument nicht zu sehen sind.

Danach sucht Mallory nach zwei Dokumenten, die denselben Hashwert besitzen. Findet Mallory ein solches Paar (die Wahrscheinlichkeit dafür ist relativ groß), so legt er das eine Dokument Alice zur Unterschrift vor. Nachdem Alice die Überweisung an Bob mit dem kleinen Betrag digital unterschrieben hat, trennt Mallory die Signatur vom Dokument ab und hängt sie unter das Dokument, in dem Alice eine große Summe an Mallory überweist.

Da die digitale Unterschrift dem Verschlüsseln des Hashwerts des Dokuments mit dem privaten Schlüssel entspricht und beide Dokumente denselben Hashwert besitzen, ist Alices Unterschrift auch auf dem ruinösen Dokument gültig. Mallory hat sich also erfolgreich eine Unterschrift von Alice erschlichen.

Fazit: Unterschreiben Sie niemals ein Dokument (auch wenn es für Sie im Klartext abgefasst ist), das Sie nicht selbst verfasst haben. Es sei denn, Sie nehmen vor der Unterzeichnung einige subtile Änderungen vor. Beispielsweise könnten Sie vor dem Unterzeichnen selbst noch Leerzeichen hinzufügen oder entfernen. Damit ändern Sie wiederum den Hashwert der Ihnen vorgelegten Daten und machen Mallorys gefährliches Zweitdokument wertlos.

11.9 Public Key Infrastructure

Was ist eine PKI?

Anhand der Probleme, die sich durch eine Man-in-the-Middle-Attacke ergeben, haben Sie gesehen, dass trotz des einfachen Schlüsselmanagements und der Verschlüsselungsfunktionen eine wesentliche Schwachstelle bei asymmetrischen Verfahren besteht. Wenn Sie mit einem anderen Teilnehmer verschlüsselt kommunizieren wollen oder auch nur dessen digitale Unterschrift überprüfen, benötigen Sie dessen Public Key. Wenn Sie diesen allerdings über eine ungesicherte Quelle (z. B. einen Key-Server aus dem Internet) beziehen, haben Sie keine Garantie, dass der öffentliche Schlüssel wirklich zu Ihrem Partner gehört.

In asymmetrischen Systemen ist deswegen die Möglichkeit vorgesehen, die **Echtheit eines Schlüssels durch eine digitale Signatur einer Zertifizierungsstelle zu bestätigen**. Im Detail könnte ein solcher Vorgang so ablaufen:

Alice möchte mit Bob kommunizieren, hat aber dessen öffentlichen Schlüssel noch nicht. Beide befürchten, dass Mallory die Daten auf dem Key-Server manipulieren könnte, um Alice und Bob seinen eigenen Schlüssel unterzuschieben ①.

Deswegen einigen Alice und Bob sich darauf, dass Bob seinen öffentlichen Schlüssel Trent aushändigt, der als Treuhänder fungiert und sich persönlich davon überzeugen kann, dass der ihm übergebene Schlüssel wirklich Bob gehört ②. Alice besitzt bereits Trents öffentlichen Schlüssel, da sie diesen persönlich von ihm bekommen hat.

Trent beglaubigt die Echtheit von Bobs Schlüssel nun durch seine digitale Unterschrift auf dem Schlüssel ③. Er stellt ihm ein sogenanntes **Zertifikat** aus. Den so signierten öffentlichen Schlüssel stellt er nun zum Download bereit.

Alice holt Bobs signierten Schlüssel und kann anhand der Unterschrift von Trent prüfen, ob sie diesen Schlüssel unmodifiziert erhalten hat ④. Wenn das der Fall ist, so benutzt Alice den Schlüssel, um Bob eine verschlüsselte Nachricht zukommen zu lassen.

Dieses Beispiel macht deutlich, dass Alice und Bob jetzt sicher miteinander kommunizieren können, ohne dass sie persönlich die Schlüssel tauschen müssen. Sie haben diese Aufgabe an einen Dritten (den Treuhänder Trent) ausgelagert, dem beide vertrauen müssen.

Aufgaben einer PKI

Die Hauptaufgaben einer Schlüsselinfrastruktur sind die Bildung von Vertrauensbeziehungen und das Ausstellen von Echtheitszertifikaten. Zwei bekannte Ansätze sind:

- OpenPGP, ein dezentraler Standard
- X.509, das zentral und hierarchisch organisiert ist

OpenPGP

Das Verschlüsselungspaket PGP (Pretty Good Privacy), das Anfang der 90er-Jahre von Phil Zimmermann als Freeware herausgegeben wurde, war die Grundlage für den OpenPGP-Standard, der von Zimmermann stark mitbeeinflusst wurde. Bezüglich der Schlüsselinfrastruktur und der Vertrauensbeziehungen gilt hier ein Konzept, das als **Web of Trust** (Vertrauensnetzwerk) bekannt ist.

Bei OpenPGP kann jeder Teilnehmer am Verschlüsselungssystem, der ein gültiges Schlüsselpaar besitzt, selbst die Echtheit anderer Schlüssel durch seine Unterschrift bestätigen. Die Idee hierbei ist, dass Personen, die die Echtheit der Schlüssel auf ihrem elektronischen Schlüsselbund (die Sammlung aller bekannten öffentlichen Schlüssel) bestätigt haben, diese Schlüssel signieren und die signierten Schlüssel wiederum öffentlich verfügbar machen.

Bezieht nun ein neuer Teilnehmer so unterschriebene öffentliche Schlüssel, kann es sein, dass ein öffentlicher Schlüssel bereits mehrere Signaturen trägt. Vertraut der Teilnehmer mindestens einem der unterzeichnenden Schlüssel, so ist der neue öffentliche Schlüssel für ihn gültig.

Jeder Teilnehmer im Web of Trust kann selbst festlegen, welchen Schlüsseln er vertrauen möchte. Dies definiert er darüber, welche Schlüssel **Trusted Introducer** sein können und welche nicht. Zusätzlich ist es auch möglich, marginales Vertrauen in bestimmte Schlüssel anzugeben, sodass zum Beispiel 2 Unterschriften auf einem neuen Schlüssel nötig sind, damit der Teilnehmer diesen Schlüssel als gültig akzeptiert.

Die Sicherheit eines Web of Trust hängt zum einen von der Disziplin der Teilnehmer ab und zum anderen von der Gründlichkeit, mit der diese sich gegenseitig ihre öffentlichen Schlüssel signieren. Würden sie ohne weitere Prüfung einen öffentlichen Schlüssel signieren, den man ihnen vorlegt, so wäre das Vertrauensnetz damit ausgehebelt, weil sich möglicherweise Dritte auf die gerade geleistete Unterschrift verlassen.

X.509

Das X.509 sieht in seiner Infrastruktur eine streng vorgegebene Hierarchie vor. Es gibt eine Zertifizierungsinstanz an der Spitze, der alle Teilnehmer vertrauen müssen: die Certificate Authority (CA). Die CA ist in der Grundidee die Einzige, die Zertifikate ausstellen kann. Wenn das Netz allerdings sehr groß ist, kann die CA auch Zertifikate ausstellen, die Sub-CAs gestatten, wiederum selbst Zertifikate auszustellen. Dies ermöglicht eine Delegation der Arbeit und den hierarchischen Aufbau des Vertrauensnetzwerkes.

Möchte Alice eine sichere Nachricht an Bob senden und erhält dazu von Bob ein Zertifikat nach X.509, so enthält dies folgende Daten: den öffentlichen Schlüssel von Bob und die Signatur der für Bob zuständigen CA, die ihm das Zertifikat ausgestellt hat.

Die CA, die Bobs Zertifikat ausgestellt hat, kennt Alice nicht, allerdings befinden sich Alice und Bob in derselben Hierarchie. Die unterzeichnende CA besitzt ebenfalls ein Zertifikat, das von der nächsthöheren CA in der Organisation ausgestellt wurde.

In diesem Beispiel ist die nächsthöhere CA diejenige an der Spitze, die sogenannte **Root-CA**. Alice kann das Zertifikat der Root-CA überprüfen, da in X.509 die Teilnehmer eines Systems der gemeinsamen Root vertrauen müssen. Alice erkennt also das Sub-CA-Zertifikat als gültig an und weiß somit, dass sie den Zertifikaten, die die Sub-CA ausstellt, ebenfalls vertrauen kann. Sie kann also Bobs Zertifikat vertrauen.

Eine PKI nach X.509 ist in vielen kommerziellen Produkten implementiert. Die bekannteste Anwendung ist die Authentifizierung eines Webservers über ein entsprechendes Zertifikat im Browser. Der Benutzer kann somit sicher sein, dass die Website, die er gerade besucht, wirklich vom angezeigten Server stammt.

Damit der Internetbrowser die Zertifikate der Websites auch überprüfen kann, sind die Zertifikate der Root-CAs entweder im Browser oder gleich schon im Betriebssystem verankert. Bekannte Root-CAs unter den vorinstallierten Browserzertifikaten sind: Deutsche Telekom, Verisign, Thawte.

Selbst erstellte Zertifikate

Ein zunehmendes Problem stellen Zertifikate dar, die Server sich selber ausstellen. Mit diesen kann auch HTTPS-Verschlüsselung betrieben werden. Sie stammen aber von keiner vertrauenswürdigen Quelle. Benutzer tendieren jedoch dazu, zunehmend auch diese unsicheren Zertifikate relativ unbedacht zu akzeptieren, wodurch die Netzwerksicherheit unterminiert wird.

11.10 Übung

Fragen zur asymmetrischen Kryptografie

Übungsdatei: -- **Ergebnisdatei: *uebung11.pdf***

1. Was ist ein Hashwert und welche Hashfunktionen werden hauptsächlich eingesetzt?
2. Was verstehen Sie unter PKI?
3. Welche Aussage trifft auf die asymmetrische Verschlüsselung zu?

a		Alle Teilnehmer teilen sich einen Schlüssel.
b		Ein Schlüsselpaar setzt sich aus einem Private Key und einem Public Key zusammen.
c		Für jeden Teilnehmer existiert ein Schlüsselpaar.
d		Der Private Key wird an alle Teilnehmer weitergereicht.
e		Der Public Key kann gefahrlos verteilt werden.
f		Es besteht keine Möglichkeit, aus dem Public Key den Private Key zu berechnen.

4. Was ist eine digitale Signatur?

a		eine Verschlüsselung
b		eine elektronische Unterschrift basierend auf der symmetrischen Verschlüsselung
c		ein Hashwert
d		eine elektronische Unterschrift basierend auf der asymmetrischen Verschlüsselung

5. Um von Public-Key-Infrastruktur (PKI) sprechen zu können, müssen folgende Funktionen zur Verfügung gestellt werden:

a		Es werden Zertifizierungsstellen, Tools für die Schlüssel- und Zertifikats-verwaltung und Anwendungen, die öffentliche Schüssel verwenden können, zur Verfügung gestellt.
b		Public-Key-Infrastruktur ist eine Kombination von Software, Verschlüsselungs-technologien und Diensten, die den Umgang mit öffentlichen Schlüsseln ermöglichen.
c		Zur Verfügung gestellt werden ausschließlich von Personen ausgestellte Zertifikate.