Kryptographie

Studienbrief 6

universität wien

Digitale Signatur und Anwendungen im Internet

Gerald und Susanne Teschl

Druckfehler/Feedback bitte an:
gerald.teschl@univie.ac.at

SS 23

Version:
2023-06-30

Studienbrief 6

Digitale Signatur und Anwendungen im Internet

Inhalt

6.1 Digitale Signatur und Authentifizierung

Neben der Verschlüsselung von Nachrichten gibt es in der elektronischen Kommunikation noch ein weiteres, mindestens genauso wichtiges Problem: Bei einem gewöhnlichen Brief ist es leicht möglich, Unversehrtheit und Authentizität zu gewährleisten. Man kann nicht einfach einen Absatz herausschneiden und durch einen anderen ersetzen oder die Unterschrift kopieren. Bei elektronischen Dokumenten wäre das aber kein Problem. Deshalb ist eine **digitale Signatur** notwendig, mit der jeder sicherstellen kann, dass

- die Daten (z.B. ein Vertrag, Software) nicht verändert wurden (**Integrität der Daten**) und
- vom angegebenen Sender (Vertragspartner, Software-Hersteller) stammen (**nichtabstreitbare Urheberschaft).**

Wir haben schon in Studienbrief 5 kurz angedeutet, dass man mit einem PKK System eine digitale Signatur verwirklichen kann, die diese Anforderungen erfüllt: Wenn Alice mit ihrem geheimen Schlüssel d einen Klartext x verschlüsselt, also

$s = D_d(x)$ berechnet und $(x, s) = (x, D_d(x))$ versendet, dann kann *alle Welt* mit Alices öffentlichem Schlüssel überprüfen ob $x = E_e(s)$ gilt. Falls ja, so ist klar, dass $s = E_e^{-1}(x) = D_d(x)$ und da nur Alice D_d kennt, kann die Nachricht nur von ihr sein. Außerdem kann die Nachricht auch nicht verändert worden sein.

Ein Nachteil des so beschriebenen Verfahrens ist, dass die übertragene Datenmenge sich verdoppeln würde, da die Signatur s etwa genauso groß wie der Klartext x wäre. Ausserdem wurde ja auch schon erwähnt, dass es ineffizient ist, größere Datenmengen mit RSA zu verschlüsseln.

Deshalb signiert man nicht das gesamte Dokument, sondern nur seinen digitalen Fingerabdruck. Ein Signaturalgorithmus besteht demnach immer aus zwei Teilen: einer (kryptographischen) Hashfunktion und einem asymmetrischen kryptographischen Algorithmus. Einer der am häufigsten verwendeten Signaturalgorithmen ist der **RSA-Signatur-Algorithmus**. Im folgenden wird die Grundidee beschrieben, der in der Praxis verwendete Standard ist in [FIPS186-5; RFC8017] beschrieben.

- H sei eine kryptographische Hashfunktion, auf die sich Bob und Alice geeinigt haben. (n, d) bzw. (n, e) seien der private bzw. öffentliche RSA-Schlüssel von Alice. x ist das Dokument, das Alice signieren möchte.
- Alice berechnet den Hashwert des Dokuments x, also $h = H(x)$, und signiert diesen Hashwert, indem sie ihn mit ihrem geheimen Schlüssel RSA-verschlüsselt: $s = h^d \pmod n$. Der Wert s heißt **Signatur**. Alice gibt (x, s) als signiertes Dokument bekannt.
- Bob entschlüsselt erstens die Signatur mit Alices öffentlichem Schlüssel, berechnet also $h_v = s^e \pmod n$. Zweitens berechnet er den Hashwert des Dokuments, $h = H(x)$. Gilt $h_v = h$, so ist die Signatur gültig (v steht für *verifizieren*).

Beispiel 6.1 (→CAS) RSA-Signaturalgorithmus
Ihr geheimer RSA-Schlüssel sei $(n, d) = (1147, 149)$, die verwendete Hashfunktion ist (einfachheitshalber) die Buchstabensumme modulo 128. Signieren Sie die Nachricht „SQUEAMISH OSSIFRAGE“. Überprüfen Sie die Signatur mit dem öffentlichen Schlüssel $(n, e) = (1147, 29)$.

Lösung zu 6.1 Die Nachricht in ASCII-Code lautet: 83, 81, 85, 69, 65, 77, 73, 83, 72, 32, 79, 83, 83, 73, 70, 82, 65, 71, 69. Der Hashwert ist $h = 83 + 81 + \cdots + 69 \bmod 128 = 115$. Somit ist die Signatur $s = 115^{149} \bmod 1147 = 210$.

Überprüfen wir nun noch die erzeugte Signatur: $h_v = 210^{29} (\bmod 1147) = 115$; $h = 83 + 81 + \cdots + 69 \bmod 128 = 115$. ■

Ein anderes Protokoll dient dazu, sich **digital zu authentifizieren**. Die Idee: Bob möchte, dass sich Alice digital ausweist. Dazu wählt er irgendeinen Text x, verschlüsselt ihn mit Alices öffentlichem Schlüssel und schickt $E_e(x)$ an Alice. Nur

Alice kann den Text mit ihrem geheimen Schlüssel entschlüsseln, und so hat Bob die Gewissheit, mit Alice zu kommunizieren, wenn Alice x zurückschickt. Alice kann sich also mit ihrem geheimen Schlüssel auch authentifizieren.

Allerdings gibt es bei diesem einfachen Protokoll noch ein Problem, insbesondere wenn Alice den RSA-Algorithmus auch zur Signatur verwendet: Mallory kann die eben beschriebene Vorgangsweise missbrauchen, um Alice dazuzubringen, einen für ihn günstigen Vertrag zu signieren. Dazu schickt Mallory einfach den Hashwert $h = H(x)$ des Vertrages x an Alice, und bittet sie, sich zu authentifizieren. Alice glaubt, dass es sich um irgendeinen sinnlosen verschlüsselten Text handelt, und wendet ihren geheimen Schlüssel auf h an, um sich zu authentifizieren. Nun hat Mallory den gewünschten Vertrag x und Alices gültige Signatur darunter, $s = D_d(h)$.
Das Protokoll zur **Authentifizierung mit RSA** muss also aus Sicherheitsgründen noch wie folgt mit einer kryptographischen Hashfunktion verfeinert werden:

- H sei eine kryptographische Hashfunktion, auf die sich Bob und Alice geeinigt haben. (n, d) bzw. (n, e) seien der private bzw. öffentliche RSA-Schlüssel von Alice. Bob möchte, dass Alice sich identifiziert.
- Bob schickt an Alice $\rho = E_e(r)$ eines zufälligen Textes r.
- Alice wendet auf ρ ihren geheimen Schlüssel an, $D_d(\rho)$, und berechnet davon den Hashwert $h = H(D_d(\rho)) = H(D_d(E_e(r))) = H(r)$, den sie an Bob zurückschickt.
- Bob berechnet ebenfalls den Hashwert des Zufallstextes, $h_v = H(r)$, und überprüft, ob $h_v = h$. Wenn ja, dann hat er sich davon überzeugt, dass er mit Alice kommuniziert.

Mallory wurde damit ein Strich durch die Rechnung gemacht, denn Mallory würde nur $H(D_d(h))$ erhalten, woraus sich $D_d(h)$ aber nicht berechnen lässt (da H eine Einweg-Hashfunktion ist).

Wenn Sie also jemand bittet, eine *Testnachricht* zu Probezwecken zu entschlüsseln und das Ergebnis bekannt zu geben, tun sie es nicht (vergleiche auch Punkt (6) in Abschnitt 5.5)! Oder treten beim Entschlüsseln Probleme auf (weil das Resultat nicht den Erwartungen entspricht: fehlerhaftes Format, kein sinnvoller Text, etc.), geben Sie keine Details bekannt! Jede Information dieser Art kann bei einem Angriff helfen.

Elgamal kann, wie wir schon wissen, nicht direkt zur Erzeugung digitaler Signaturen verwendet werden.

Denn Alice braucht zur Verschlüsselung mit Elgamal nicht nur ihren geheimen Schlüssel a, sondern – zur Berechnung von k – auch Bobs öffentlichen Schlüssel β. Damit würde die Signatur auch eine (zwar öffentlich bekannte) Zutat von Bob enthalten und wäre nur für Bob überprüfbar.

Es gibt daher eine Variante zur Erzeugung einer Signatur mit dem privaten Schlüssel von Elgamal, den **Elgamal-Signaturalgorithmus**:

- Es sei p eine Primzahl und $g \in \mathbb{Z}_p^*$. H sei eine kryptographische Hashfunktion, auf die sich Bob und Alice geeinigt haben. $a \in \mathbb{Z}_{p-1}$ bzw. $\alpha = g^a$ seien der private bzw. öffentliche Elgamal-Schlüssel von Alice. Alice möchte das Dokument x signieren.

- Alice wählt eine (geheime) Zufallszahl $r \in \mathbb{Z}_{p-1}^*$ (**Nonce**) und berechnet $\rho = g^r \pmod p$ und das multiplikative Inverse $r^{-1} \in \mathbb{Z}_{p-1}^*$. Weiters berechnet Alice $h = H(x)$ und $s = (r^{-1}(h - a\rho)) \pmod{(p-1)}$. Ist $s \neq 0$ so wird x gemeinsam mit (ρ, s) als Signatur bekanntgegeben. Ansonsten wird der Vorgang mit einem neuen r wiederholt.

- Bob berechnet $v_1 = \alpha^\rho \rho^s \pmod p$ und $v_2 = g^{H(x)} \pmod p$. Gilt $v_1 = v_2$, so ist die Signatur echt.

Wegen $rs = h - a\rho \pmod{(p-1)}$ gilt $v_1 = (g^a)^\rho (g^r)^s = g^{a\rho + rs} = g^h$, also ist der Algorithmus korrekt.

Folgende Bemerkungen zur Sicherheit sind zu beachten:

Die Zufallszahl r muss (kryptographisch sicher erzeugt werden) und geheim bleiben (z.B. nach dem Berechnen der Signatur vernichtet), denn sonst kann der geheimen Schlüssel via $a = \rho^{-1}(h - rs)$ berechnet werden (ist $s = 0$, so braucht man r nicht, und deshalb muss dieser Fall ausgeschlossen werden). Umgekehrt sehen wir, dass bei Kenntnis von a die Zufallszahl r durch Lösen von $rs = h - a\rho$ bestimmt werden kann. Die Berechnung von a aus s ist also genauso schwer wie die Berechnung von r aus ρ (also das Lösen des DLP $\rho = g^r$).

Um eine gültige Signatur ohne Kenntnis von a zu erzeugen, müsste Eve Zahlen (ρ, s) mit der Eigenschaft $\alpha^\rho \rho^s = g^h$ finden. Der naive Zugang wäre es, ρ zu wählen. Dann muss aber das DLP $\rho^s = g^h \alpha^{-\rho}$ gelöst werden. Auf den ersten Blick erscheint das einfach indem man ein ρ mit kleiner Ordnung wählt, aber dann erzeugt ρ auch nur eine kleine Untergruppe und es ist unwahrscheinlich, dass $g^h \alpha^{-\rho} \in \langle \rho \rangle$ ist (das DLP wird also in der Regel unlösbar sein).

Außerdem darf r nur einmal verwendet werden! Ansonsten kann Mallory aus $s_1 = r^{-1}(h_1 - a\rho)$ und $s_2 = r^{-1}(h_2 - a\rho)$ zunächst r aus $(s_1 - s_2)r = (h_1 - h_2)$ berechnen und daraus dann $a = \rho^{-1}(h_1 - rs_1)$ erhalten.

Sony hat diesen Fehler der Mehrfachverwendung von r bei der PlayStation 3 gemacht, wodurch der geheime Schlüssel, mit dem Sony die Spiele signiert, bekannt wurde.

Beispiel 6.2 Elgamal-Signaturalgorithmus
Die Systemparameter seien $(p, g) = (167, 4)$. Ihr geheimer Elgamal-Schlüssel sei $a = 21$ und wieder wird die Ziffernsumme modulo 128 als Hashfunktion verwendet. Signieren Sie die Nachricht „SQUEAMISH OSSIFRAGE“. Überprüfen Sie die Signatur mit dem öffentlichen Schlüssel $(p, g, \alpha) = (167, 4, 154)$.

Lösung zu 6.2 Den Hashwert $h = 115$ haben wir bereits im letzten Beispiel berechnet. Unser Schlüssel ist $a = 21$ und unsere geheime Zufallszahl sei $r = 31$. Nun berechen wir $\rho = 4^{31} (\mathrm{mod}\, 167) = 54$. Das zugehörige Inverse modulo 166 ist $r^{-1} = 75$. Damit ergibt sich $s = r^{-1}(h - \rho a) (\mathrm{mod}\, p-1) = 101$ und die Signatur ist $(\rho, s) = (54, 101)$. Überprüfen wir nun die erzeugte Signatur: Mit $\alpha = g^a (\mathrm{mod}\, p) = 154$ erhalten wir $\alpha^\rho \cdot \rho^s (\mathrm{mod}\, p) = 49 = g^h (\mathrm{mod}\, p)$. ■

Die Tatsache, dass die Elgamal-Signatur eine (kryptographisch sichere) Zufallszahl benötigt, ist ein Nachteil gegenüber der RSA-Signatur, die allerdings umgangen werden kann, indem man r über einen Hashwert aus der Nachricht und dem privaten Schlüssel generiert [RFC6979].

Das NIST hat 1991 nicht RSA sondern eine Variante von Elgamal, den **Digital Signature Algorithm** (DSA) zum **Digitial Signature Standard** (DSS) gemacht.

Die Zahl g wird bei DSA als Generator einer Untergruppe mit wesentlich kleinerer Ordnung $m = \mathrm{ord}(g)$ als p festgelegt und anstelle $(\mathrm{mod}\,(p-1))$ wird $(\mathrm{mod}\, m)$ gerechnet. Das ist natürlich immer möglich und hat den Vorteil, dass es zu kleineren Signaturen führt. Da die Laufzeit des Index Calculus von p (und nicht von m) abhängt, reicht es, wenn das DLP in der Untergruppe vor den generischen Verfahren (Baby Step-Giant Step bzw. Pollard Rho) sicher ist. Ist ausserdem m prim, dann muss man bei der Wahl von r nicht testen ob es teilerfremd zu m ist.

Die genauen Gründe dafür sind nicht bekannt, aber hängen einerseits wohl mit patentrechtlichen Überlegungen als auch mit der Tatsache zusammen, dass DSA nur zum Signieren, und nicht wie RSA auch zum Verschlüsseln, verwendet werden kann. Für Verschlüsselungsverfahren gab es damals nämlich noch restriktive Exportbeschränkungen (vgl. dazu auch die Geschichte von PGP in Abschnitt 6.3).

Ein Nachteil von DSA gegenüber RSA ist, dass das Überprüfen einer Signatur wesentlich länger dauert. Ein Vorteil ist, dass DSA auf einer öffentlichen Primzahl p beruht, wodurch ein Betrug durch absichtlich schlecht gewähltes p verhindert wird.

DSS wurde zuletzt 2023 angepasst [FIPS186-5] und umfasst nun auch den RSA Signaturalgorithmus sowie den **ECDSA** (**Elliptic Curve Digital Signature Algorithm**) und **EdDSA** (**Edwards-Curve Digital Signature Algorithm**). Letztere basieren auf der Beobachtung, dass man $\mathbb{Z}_p$, wie bereits mehrfach erwähnt, durch eine beliebige andere Gruppe ersetzten kann. Der DSA wird nicht mehr empfohlen.

Zum Schluß erwähnen wir noch eine elegante Variante, die **Schnorr-Signatur** nach dem deutschen Mathematiker Claus-Peter Schnorr (*1943).

- Es sei p eine Primzahl und $g \in \mathbb{Z}_p^*$ mit primer Ordnung $m = \mathrm{ord}(g)$. H sei eine kryptographische Hashfunktion, auf die sich Bob und Alice geeinigt haben. $a \in \mathbb{Z}_m$ bzw. $\alpha = g^a$ seien der private bzw. öffentliche Elgamal-Schlüssel von Alice. Alice möchte das Dokument x signieren.

- Alice wählt eine (geheime) Zufallszahl $r \in \mathbb{Z}_m^*$ (**Nonce**) und berechnet $\rho = g^r$

(mod p). Weiters berechnet Alice $h = H(\rho\|x) \pmod m$ und $s = r + ah \pmod m$ und gibt (s, h) als Signatur bekannt.

- Bob berechnet $\rho_v = g^s\alpha^{-h}$ und $h_v = H(\rho_v\|x)$. Gilt $h_v = h$, so ist die Signatur echt.

Wegen $\rho_v = g^s\alpha^{-h} = g^{r+ah-ah} = g^r = \rho$ ist der Algorithmus korrekt.

Als Signatur könnte auch (ρ, s) bekannt gegeben werden (dann berechnet man $h_v = H(\rho\|x)$ und überprüft ob $\rho_v = \rho$), die ist zwar größer, aber man kann dann mehrere Signaturen auf einmal prüfen, indem die Potenzgesetzte verwendet. Das ist bei Anwendungen, bei denen viele Signaturen geprüft werden müssen (z.B. Bitcoin), nützlich.

Wie bei Elgamal, kann bei Mehrfachverwendung von r der geheime Schlüssel aus $s_1 - s_2 = a(h_1 - h_2)$ berechnet werden. Dass ρ in die Berechnung von h einfließt ist wichtig, ansonsten könnte man beim mehrfachen Signieren der gleichen Nachricht mit verscheidenen r den geheimen Schlüssel berechnen.

Beispiel 6.3 Schnorr-Signaturalgorithmus
Die Systemparameter seien $(p, g, m) = (167, 4, 83)$. Ihr geheimer Schnorr-Schlüssel sei $a = 21$, wieder wird die Ziffernsumme modulo 128 als Hashfunktion verwendet. Signieren Sie die Nachricht „SQUEAMISH OSSIFRAGE“. Überprüfen Sie die Signatur mit dem öffentlichen Schlüssel $(p, g, \alpha) = (167, 4, 154)$.

Lösung zu 6.3 Wie zuvor ist $p = 167$ und $g = 4$. Die Ordnung von g ist die Primzahl 83. Unser geheimer Schlüssel ist $a = 21$ und unsere geheime Zufallszahl sei $r = 31$. Nun berechen wir $\rho = 4^{31} (\mathrm{mod}\, 167) = 54$, wandeln das in einen String um, den wir vor die Nachricht hängen, und den Hashwert modulo m berechnen:

$$h = H(\text{„}54\text{SQUEAMISH OSSIFRAGE“}) \bmod 83 = 92 \bmod 83 = 9.$$

Damit ergibt sich $s = r + ah \bmod 83 = 54$ und die Signatur ist $(s, h) = (54, 9)$. Überprüfen wir nun die erzeugte Signatur: Mit $\rho_v = g^s\alpha^{-h} = 54$ erhalten wir $h_v = 9 = h$.

Gibt man alternativ $(\rho, s) = (54, 54)$ bekannt, so folgt $h_v = 9$ und es gilt $\rho_v = g^s\alpha^{-h} = 54 = \rho$. ■

6.2 Authentizität öffentlicher Schlüssel und Zertifizierung

Zuletzt müssen wir noch auf ein prinzipielles Problem von Public-Key-Verfahren eingehen. Es scheint zwar so, als ob Public-Key-Verfahren das Problem der sicheren

Kommunikation über ein öffentliches Medium lösen. Das stimmt aber nur so lange, wie ein Angreifer die Verbindung nur (passiv) belauschen, nicht aber (aktiv) in die Verbindung eingreifen kann. Denn dann ist folgender Betrug möglich:

Angenommen, Alice und Bob möchten verschlüsselt miteinander kommunizieren. Dazu schickt Alice ihren öffentlichen Schlüssel e_A an Bob und Bob schickt umgekehrt seinen öffentlichen Schlüssel e_B an Alice:

$$A \quad \overset{e_A}{\underset{e_B}{\rightleftarrows}} \quad B$$

Wenn es nun einem Angreifer Mallory gelingt, sich in die Mitte der Verbindung zu setzen, so kann er beiden Seiten vortäuschen, jeweils der andere zu sein: Mallory fängt den Schlüssel e_A von Alice ab, und schickt stattdessen seinen eigenen öffentlichen Schlüssel e_M an Bob. Analog verfährt er mit dem Schlüssel von Bob:

$$A \quad \overset{e_A}{\underset{e_M}{\rightleftarrows}} \quad M \quad \overset{e_M}{\underset{e_B}{\rightleftarrows}} \quad B$$

Von nun an verschlüsselt Alice jede Nachricht an Bob mit Mallorys öffentlichen Schlüssel e_M. Mallory entschlüsselt die Nachricht mit seinem privaten Schlüssel d_M, liest sie (und verändert sie vielleicht sogar), verschlüsselt sie wieder mit Bobs öffentlichem Schlüssel e_B, und schickt sie weiter an Bob. Analoges passiert mit Nachrichten von Bob an Alice. Mallory kann also die ganze Kommunikation abhören und Nachrichten manipulieren, ohne dass Alice oder Bob etwas davon bemerken. Dieser Angriff ist als **Man-in-the-Middle-Angriff** bekannt.

Nicht nur bei der verschlüsselten Kommunikation, sondern auch beim Überprüfen einer digitalen Signatur muss man zuvor sicherstellen, dass man den richtigen öffentlichen Schlüssel verwendet. Ebenso ist der Diffie Hellman Schlüsselaustausch (Mallory tauscht α von Alice und β von Bob gegen sein eigenes $\mu = g^m \bmod p$ aus) verwundbar in Bezug auf einen Man-in-the-Middle-Angriff (deshalb wird nach dem Austausch des Sitzungsschlüssels, dieser von beiden Seiten digital signiert und die Signaturen verglichen, was als **Station-to-Station Protokoll** bekannt ist).

Gerade bei der Kommunikation über das Internet kann dieser Angriff nicht ausgeschlossen werden. Es gibt jedoch verschiedene Möglichkeiten, um diesen Angriff zu verhindern:

a) Im einfachsten Fall vergleichen Alice und Bob die erhaltenen öffentlichen Schlüssel über einen **unabhängigen Kanal**, für den es unwahrscheinlich ist, dass er auch von Mallory kontrolliert wird (zum Beispiel rufen sie sich an). Da die öffentlichen Schlüssel zu lang sind, werden in der Praxis nur Hashwerte der öffentlichen Schlüssel (digitale Fingerabdrücke) verglichen, die ja die Schlüssel eindeutig identifizieren. Diese Lösung ist aber sicher nicht immer möglich bzw. zu umständlich.

Beispiel Secure Shell (ssh): Wenn Alice sich zum ersten Mal mit dem Server verbindet, bekommt sie den Fingerabdruck des öffentlichen Schlüssels des Servers. Es ist ihr überlassen, diesen Schlüssel per Telefon oder auf anderem Weg zu überprüfen, oder gleich zu akzeptieren.

b) Eine andere Möglichkeit ist, dass eine dritte Person, Trent, der Alice und Bob vertrauen, die Echtheit der öffentlichen Schlüssel garantiert. Mehr dazu weiter unten (siehe **Zertifizierungsstelle** oder **Web of Trust**).

c) Eine weitere Möglichkeit ist das sogenannte **Interlock-Protokoll**, erstmals vorgeschlagen von R. Rivest und A. Shamir [RS84]: Alice und Bob tauschen ihre Schlüssel aus. Dann schickt Alice die erste Hälfte (jedes zweite Bit) ihrer verschlüsselten Nachricht an Bob. Erst nachdem diese Nachricht angekommen ist, schickt Alice die zweiten Hälfte ihrer Nachricht. Analog würde Bob vorgehen, wenn er eine Nachricht an Alice sendet. Wozu? – Hat Mallory sich dazwischen geschalten, so kann er die erste Hälfte der verschlüsselten Nachricht nicht entschlüsseln (diese halbe Nachricht kann von *niemandem*, auch nicht von Bob, enschlüsselt werden). Erst wenn Bob den Erhalt der ersten Hälfte bestätigt hat, schickt Alice die zweite Hälfte. Es gibt aber doch einige praktische Probleme für die Umsetzung des Interlock-Protokolls, daher kommt es kaum zu Anwendung.

Erstens brauchen Alice und Bob einen unabhängigen Kanal, über den Bob Alice mitteilen kann, wann die erste Hälfte angekommen ist (z.B. ruft Bob dann an). Wenn es so einen Kanal aber gibt, dann könnten die beiden gleich über diesen Kanal sicherstellen, dass sie den richtigen öffentlichen Schlüssel des anderen haben.

Zweitens kann Mallory – nachdem er, wie beim Man-in-the-Middle-Angriff, seinen öffentlichen Schlüssel geschickt hat – die erste Hälfte von Alice abfangen; dann verschlüsselt er eine erfundene Nachricht mit seinem eigenen öffentlichen Schlüssel und schickt die erste Hälfte davon an Bob. Dann schickt Alice die zweite Hälfte ihrer Nachricht, Mallory fängt sie wieder ab und schickt die zweite Hälfte seiner (erfundenen) Nachricht an Bob. Nun kann Mallory die gesamte Nachricht von Alice entschlüsseln, und Bob entschlüsselt die Nachricht von Mallory. Alice und Bob müssten in diesem Fall erkennen können, dass die Nachrichten ersetzt wurden (Mallory hätte aber in jedem Fall bis zur Enttarnung die Nachrichten abgehört).

Möchte man die Überprüfung nicht dem Benutzer übertragen (siehe a) oben), so benötigt man einen vertrauenswürdigen Dritten, der die Echtheit von öffentlichen Schlüsseln überprüft und dann durch seine digitale Signatur bestätigt (siehe b) oben).

Man bezeichnet eine solche zentrale Beglaubigungsinstanz als **Zertifizierungsstelle** bzw. **Certification Authority** (**CA**) (oder auch *Trust Center (TC), Trusted Third Party (TTP)*). Das läuft folgendermaßen ab:

- Es wird ein Dokument mit der Benutzeridentifikation (z.B.: vollständiger Name und Geburtsdatum, Emailadresse oder Domainname eines Servers) und dem zugehörigen öffentlichen Schlüssel erstellt (= Zertifizierungsantrag, *Certification Signing Request (CSR)*).

- Dieses Dokument wird dann der CA vorgelegt, die es nach sorgfältiger Überprüfung aller Daten (zusammen mit einem Verweis auf den Aussteller) digital signiert.

- Dieses Dokument, zusammen mit der Signatur der CA, ist das **Zertifikat**, das von nun an die Benutzeridentifikation und den öffentlichen Schlüssel fälschungssicher miteinander verbindet. Da jeder Benutzer den öffentlichen Schlüssel der CA kennt, kann er die Signaturen und somit die Zertifikate der CA überprüfen.

- Zertifikate haben immer ein Ablaufdatum und zusätzlich führt die CA eine Liste (**Certificate Revocation List**, CRL) mit der sie Zertifikate nachträglich für ungültig erklären kann (falls z.B. der geheime Schlüssel des Anwenders kompromittiert wurde).

Dieses System, bei der es eine kleine Anzahl zentraler Stellen gibt, die Zertifikate ausstellen, wird zum Beispiel bei SSL bzw. dessen Nachfolger TLS (Industriestandard für verschlüsselte Internetverbindungen) und S/MIME (Industriestandard für verschlüsselte Emails) verwendet.

Die öffentlichen Schlüssel vieler Zertifizierungsstellen (die zum Überprüfen der Zertifikate notwendig sind) sind dabei in der Regel schon in die Software (z.B. Webbrowser) integriert (und können ggf. durch eigene ergänzt werden). Da bei diesem System alle Benutzer darauf vertrauen, dass die Zertifizierungsstelle Personen und öffentliche Schlüssel korrekt einander zuordnet, ist sie ein essentieller Bestandteil der Sicherheit des gesamten Systems und trägt somit eine hohe Verantwortung. Um eine digitale Signatur einer CA für den eigenen öffentlichen Schlüssel zu bekommen, wird also ein Webformular nicht ausreichen, sondern man wird z.B. persönlich mit einem Identifikationsnachweis (Reisepass, etc.) vorsprechen müssen. Es ist klar, dass damit für die Zertifizierungsstelle ein hoher Aufwand und somit Kosten verbunden sind.

Diese Kosten sind aber nicht in jedem Fall erwünscht, deshalb gibt es alternativ auch eine private (dezentrale) Lösung: *Jeder* Benutzer kann andere Schlüssel zertifizieren und entscheidet für sich, von welchen Benutzern er Zertifikate akzeptiert. Zum Beispiel überprüft und signiert Trent den öffentlichen Schlüssel von Bob. Alice vertraut dem Zertifikat von Trent (also darauf, dass Trent zuverlässig überprüft hat, und darauf, dass die Signatur von Trent echt ist), und akzeptiert daher ebenfalls den Schlüssel von Bob. Dieses Modell wird beim **Web of Trust** in PGP verwendet. Für den Benutzer fallen zwar keinerlei Kosten für Zertifikate an, er muss aber selbst entscheiden, von welchen Benutzern er Zertifikate akzeptiert.

Schlüsselerzeugung und CSR mit OpenSSL

Wollen sie im Internet einen verschlüsselten Serverdienst (Z.B. Webserver — https) betreiben oder ihre Emails verschlüsseln/signieren, so benötigen Sie also ein Schlüsselpaar und ein Zertifikat.

Einen privaten Schlüssel und den Zertifizierungsantrag kann man sich mit der Software OpenSSL (`openssl.org`) erstellen. Die meisten Zertifizierungsstellen er-

zeugen auf Wunsch natürlich den Schlüssel gleich mit, aber als paranoide Sicherheitsfanatiker wollen wir nicht, dass unser geheimer Schlüssel unseren Rechner verlässt.

Einen RSA-Schlüssel erzeugt man z.B. mit[1]

```
[tux@soliton tux]$ stty -echo; openssl genrsa -aes128 -passout stdin
 -out rsa-key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
...........+++
e is 65537 (0x10001)
```

Der private Schlüssel wird in der Datei `rsa-key.pem` abgelegt und mit AES128 verschlüsselt. Das Passwort wird zu Beginn über die Tastatur (`stdin`) eingegeben (dabei verhindert `stty -echo`, dass die Eingabe angezeigt wird;-). Die Datei ist im PEM-Format (steht für *Privacy Enhanced Mail*), ein menschenlesbares Format, das für den Versand kryptographischer Bestandteile per Email entwickelt wurde.

Alternativ kann man sich mit[2]

```
[tux@soliton tux]$ openssl ecparam -name secp384r1 -genkey
-out ec-key.pem -noout
```

auch einen DH-Schlüssel auf einer elliptischen Kurve erstellen (man beachte den Zeitunterschied bei der Erstellung). Bei Verwendung einer elliptischen Kurve gibt es keine Option die Datei mit dem Schlüssel mit AES zu verschlüsseln. Tux hätte gerne eine andere Kurve gewählt, aber die NIST-Kurven `secp256k1` und `secp384r1` bieten maximale Kompatibilität.

Den CSR erzeugt man mit

```
[tux@soliton tux]$ openssl req -new -key rsa-key.pem -out cert.csr
Enter pass phrase for rsa-key.pem:
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN. There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:AQ
State or Province Name (full name) []:
Locality Name (eg, city) []:McMurdo
```

[1] https://wiki.openssl.org/index.php/Command_Line_Utilities

[2] https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations

```
Organization Name (eg, company) []:Aptenodytes
Organizational Unit Name (eg, section) []:Forsteri
Common Name (eg, fully qualified host name) []:www.southpole.edu
Email Address []:tux@southpole.edu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: a-n-c-h-0-v-y
```

Die *pass phrase* die zu Beginn abgefragt wird, ist das Passwort das Tux bei der Schlüsselerstellung gewählt hat. Das *challenge password* ist optional und wird von der CA gespeichert um Tux bei späteren Anfragen einfach authentifizieren zu können (Achtung: es wird im Klartext im CSR abgespeichert). Der Antrag ist nun in der Datei `cert.csr` und kann bei einer Zertifzierungsstelle vorgelegt werden. Als Antwort bekommen sie ihr Zertifikat `certs.pem` wieder im PEM Format. Das kannTux nun zusammen mit dem privaten Schlüssel in die Konfiguration seines Servers einfügen.

Alternativ kann man sein Zertifikat selbst unterschreiben, dann wird in der Regel gewarnt und zum händischen Akzeptieren des Schlüssels aufgefordert. Oder man richtet sich mit OpenSSL seine eigene Zertifizierungsstelle ein. Dazu muss dann aber auch jeder Benutzer- den öffentlichen Schlüssel dieser neuen Zertifizierungsstelle bei seiner Software hinzufügen. Kostenlose Zertifikate für Webserver sind von der Non-Profit-Organisation Let's Encrypt (`letsencrypt.org`) erhältlich.

Will Tux das Zertifikat für S/MIME verwenden, dann muss er noch seinen privaten Schlüssel hinzufügen und alles in eine verschlüsselte Datei nach dem PKC12 Standard verpacken:

```
[tux@soliton tux]$ openssl pkcs12 -export -inkey rsa-key.pem
-in certs.pem -out certs.p12
```

Die Datei `cert.p12` kann Tux nun in seinen Email-Client importieren. Falls man das Erstellen des Schlüssels der Zertifizierungstelle überlassen hat, bekommt man Schlüssel und Zertifikat in diesem Format.

TLS

Nun können wir auch verstehen wie die verschlüsselte Kommunikation zwischen Client und Server nach **TLS** (*transport layer security*) abläuft. Wir schränken uns dabei auf die aktuelle Version TLS 1.3 ein, bei der die meisten veralteten Verfahren gestrichen wurden und nur noch ein DH Schlüsseltausch mit Einmalschlüssel (*ephemeral key*) vorgesehen ist. Das wird auch als DHE bezeichnet und zur Unterscheidung spricht man bei der Verwendung von Dauerschlüsseln von *static* RSA bzw. DH.

Das Aushandeln des Sitzungsschlüssels wird als **Handshake** bezeichnet und läuft in drei Schritten ab:

- **ClientHello**: Der Client schickt seinen öffentlichen DH-Schlüssel (inkl. dem Namen der verwendeten Gruppe), eine zufällige Nonce und (optional) eine Liste an symmetrischen Chiffren bzw. Hashfunktionen (HMAC), die er unterstützt, an den Server.

- **ServerHello**: Der Server berechnet das geteilte Geheimnis (mit dem er bereits seine Antwort, bis auf seinen öffentlichen Schlüssel, verschlüsselt) und wählt die symmetrische Chiffre bzw. Hashfunktion aus.

  Nun schickt er seinen öffentlichen DH-Schlüssel, eine zufällige Nonce, sein Zertifikat und seine Signatur über den gesamten bisherigen Handshake an den Client. Zusätzlich könnte er auch noch ein Zertifikat vom Client fordern.

- Der Client berechnet das geteilte Geheimnis, entschlüsselt die Antwort und überprüft das Zertifikat (Signatur, Gültigkeitsdauer, CRL). Falls verlangt, signiert auch der Client den Handshake und schickt sein Zertifikat an den Server. Ansonsten wird der gesamte Handshake mit einer Prüfziffer (Hash) signiert.

War der Handshake erfolgreich, so haben beide Seiten einen gemeinsamen Schlüssel für ein symmetrisches Verfahren mit dem alle weiteren Datenpakete der Verbindung verschlüsselt werden. Das vorgesehene Minimum, das alle Teilnehmer unterstützen müssen, ist z.B. „TLS_AES_128_GCM_SHA256“, also AES128 im Galios Counter Mode und SHA256 als Hashfunktion. Bei späteren Verbindungen, ist es vorgesehen, dass man auf die zuvor vereinbarten Schlüssel (PSK, *pre shared key*) zurückgreifen kann und dadurch nur ein vereinfachter Handshake beim Verbindungsaufbau notwendig ist.

6.3 PGP

Als einer der Pioniere der privaten Verschlüsselung veröffentlichte **Phil Zimmermann** 1991 die Software **Pretty Good Privacy** (**PGP**). Es war eine Verschlüsselungssoftware für die breite Öffentlichkeit: kostenlos, benutzerfreundlich und auf jedem Heim-PC lauffähig. Obwohl PGP als Freeware gratis zur Verfügung stand, hatte es doch zwei Schönheitsfehler: Erstens verwendete PGP die *patentierten* Algorithmen RSA und IDEA und konnte somit nur für private Zwecke gratis genutzt werden. Zweitens konnte es aufgrund der restriktiven Exportbestimmungen der USA nicht exportiert werden. Um PGP auch außerhalb der USA (legal) verwenden zu können, wurde eine Lücke im Exportgesetz genutzt und der Quellcode von PGP wurde in Buchform exportiert. Seit 1997 gibt es daher PGP auch in der

internationalen Version PGPi. Inzwischen wurden die Exportbeschränkungen aufgehoben und auch das Patent für den RSA-Algorithmus ist abgelaufen, allerdings ist PGP seit Version 9 nur mehr kommerziell verfügbar. Aus diesem Grund gibt es eine Open-Source-Variante, den **GNU Privacy Guard** (**GnuPG**).

Um die Kompatibilität der verschiedenen Implementationen zu gewährleisten wurde der OpenPGP-Standard definiert. Da verschiedene Versionen verschiedene Algorithmen zum Signieren/Verschlüsseln/Komprimieren verwenden, ist das natürlich nur eingeschränkt möglich. Insbesondere verwenden die originalen PGP Versionen (bis 2.6) den RSA-Algorithmus, während spätere PGP Versionen (seit 5.0) und GnuPG den DSS und Elgamal verwenden.

Wir wollen uns im Folgenden auf GnuPG konzentrieren, das von der offiziellen Webseite www.gnupg.org bezogen werden kann.

Nach der Installation muss als erstes ein privater und öffentlicher Schlüssel erzeugt werden. Das geschieht mit dem Befehl `gpg -gen-key`. Genaugenommen werden dabei jeweils zwei Schlüssel erzeugt: Ein Schlüsselpaar zum Signieren mit dem DSA und ein Schlüsselpaar zum Verschlüsseln nach dem Elgamal-Verfahren. Außerdem wird zum Schlüssel noch Ihre Identität (Name, Email, Kommentar) hinzugefügt.

Nach der Schlüsselerzeugung werden Sie zur Eingabe einer **Passphrase** aufgefordert - aus Sicherheitsgründen wird Ihr privater Schlüssel nur verschlüsselt (mit einem konventionellen Algorithmus) auf der Festplatte abgelegt. Nachteil davon: vor jeder Operation, die Ihren privaten Schlüssel verwendet, werden Sie zunächst nach Ihrer Passphrase gefragt werden. Vorteil: wenn jemand unbefugten Zugriff auf Ihre Daten erhält, ist Ihr privater Schlüssel trotzdem sicher (und damit auch alle damit verschlüsselten Daten). Ob letzte Aussage wirklich zutrifft, hängt allerdings von der Güte Ihrer Passphrase ab. Denn ein einfaches Passwort aus acht Zeichen (oder weniger) bietet natürlich keinen ausreichenden Schutz gegen einen Brute-Force-Angriff auf die Passphrase.

Nun können Sie sich ansehen, welche Schlüssel Sie auf Ihrem öffentlichen Schlüsselbund (*keyring*)

```
[tux@soliton tux]$ gpg --list-keys
/home/tux/.gnupg/pubring.gpg
----------------------------
pub  1024D/BC39CEE7 2003-03-09 Tux the Penguin <tux@southpole.edu>
sub  1024g/6E8E40D2 2003-03-09
```

bzw. privaten Schlüsselbund

```
[tux@soliton tux]$ gpg --list-secret-keys
/home/tux/.gnupg/secring.gpg
----------------------------
sec  1024D/BC39CEE7 2003-03-09 Tux the Penguin <tux@southpole.edu>
ssb  1024g/6E8E40D2 2003-03-09
```

haben. Sie sehen, dass Sie (= Tux) den öffentlichen bzw. privaten Hauptschlüssel (der Länge 1024 Bit) mit KeyID `BC39CEE7` zur Signatur mit DSS haben; und den

öffentlichen und privaten Unterschlüssel mit KeyID `6E8E40D2` zum Verschlüsseln mit Elgamal. Wenn Tux bereits öffentliche Schlüssel anderer Benutzer hätte, so würden diese ebenfalls auf seinem öffentlichen Schlüsselbund aufscheinen; analog, wenn er weitere private Schlüssel besitzen würde.

Als nächstes muss Tux seinen öffentlichen Schlüssel bekanntmachen. Dazu exportiert er seinen öffentlichen Schlüssel:

```
[tux@soliton tux]$ gpg -a --export tux
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.1 (GNU/Linux)

mQGiBD5rcSgRBACN/K97rAYjD8CzM7EZdufixsASfl6h9G3o9K77r1LMebt7T4L5
.....
SePQAKCJckVyZJKTC+ic3tYanljDWq9Caw==
=q5ge
-----END PGP PUBLIC KEY BLOCK-----
```

Die Option `-a` bewirkt, dass die Ausgabe nicht binär sondern im (7 Bit) ASCII-Format erfolgt. Diesen Textblock (der Ihren öffentlichen Schlüssel und weitere Informationen, wie Ihre Benutzeridentifikation und etwaige Signaturen von anderen Benutzern, enthält) können Sie nun an alle Personen schicken, mit denen Sie verschlüsselt kommunizieren wollen bzw. auf Ihre Homepage stellen. Umgekehrt müssen Sie alle öffentlichen Schlüssel, die Sie bekommen, mittels

```
[tux@soliton tux]$ gpg --import pubkey.txt
```

importieren (die Datei `pubkey.txt` enthält den Key-Block). Die Schlüssel werden dann zu Ihrem Schlüsselbund hinzugefügt.

Nun können Sie auch schon verschlüsseln, zum Beispiel das File test.txt (wenn Sie es zum Beispiel verschlüsselt auf der Festplatte ablegen möchten):

```
[tux@soliton tux]$ gpg -r tux -e test.txt
```

Mit der Option `-r tux` wählt Tux seinen eigenen öffentlichen Schlüssel aus (Sie können dazu einen Teil der Emailadresse, des Namens oder die KeyID des Benutzers angeben, dessen öffentlichen Schlüssel Sie verwenden möchten). Der Befehl erzeugt die Datei `test.txt.gpg`, die mit dem Befehl

```
[tux@soliton tux]$ gpg test.txt.gpg
You need a passphrase to unlock the secret key for
user: "Tux the Penguin <tux@southpole.edu>"
1024-bit ELG-E key, ID 6E8E40D2, created 2003-03-09 (main key ID BC39CEE7)

gpg: encrypted with 1024-bit ELG-E key, ID 6E8E40D2, created 2003-03-09
      "Tux the Penguin <tux@southpole.edu>"
```

wieder entschlüsselt werden kann. Der Befehl erzeugt wieder die entschlüsselte Originaldatei `test.txt`.

Wenn Tux wie in unserem Beispiel eine Datei mit seinem eigenen Schlüssel verschlüsselt um sie vor fremden Blicken auf seine Festplatte zu schützen, dann muss er natürlich auch die unverschlüsselte Originaldatei löschen. Für das spurenlose Löschen gibt es spezielle Software, da die gelöschte Datei ansonsten unter Umständen wiederhergestellt werden kann.

Analog können Sie auch eine Datei signieren:

```
[tux@soliton tux]$ gpg -s test.txt

You need a passphrase to unlock the secret key for
user: "Tux the Penguin <tux@southpole.edu>"
1024-bit DSA key, ID BC39CEE7, created 2003-03-09
```

Nachdem Sie die Passphrase für Ihren privaten Schlüssel eingegeben haben, wird wieder eine Datei `test.txt.gpg` erzeugt, die nun die Informationen der ursprünglichen Datei inklusive der Signatur (und die KeyID des zugehörigen öffentlichen Schlüssels, mit dem die Signatur geprüft werden kann) enthält. Jeder, der den zugehörigen öffentlichen Schlüssel hat, kann mit dem Befehl

```
[tux@soliton tux]$ gpg test.txt.gpg
gpg: Signature made Sat 15 Mar 2003 11:44:44 AM CET
using DSA key ID BC39CEE7
gpg: Good signature from "Tux the Penguin <tux@southpole.edu>"
```

die Signatur überprüfen und gleichzeitig wird auch die Datei `test.txt` erzeugt. Dies hat den Nachteil, dass, um an den Inhalt von `test.txt` zu kommen, der öffentliche Schlüssel notwendig ist. Deshalb gibt es auch die Möglichkeit, die Signatur von der eigentlichen Nachricht zu trennen: `gpg -b test.txt` (*detached signature*). In diesem Fall wird eine Datei `test.txt.sig` erzeugt, die *nur* die Signatur enthält. Zum Überprüfen sind nun natürlich beide Dateien nötig: `gpg -verify test.txt.sig test.txt`. Falls Sie den notwendigen öffentlichen Schlüssel noch nicht auf Ihrem Schlüsselbund haben, so erhalten Sie eine Warnung, aus der hervorgeht, welcher Schlüssel notwendig ist (seine KeyID wird angegeben). Diesen Schlüssel müssen Sie sich dann erst besorgen.

Die erzeugten `*.gpg` Dateien sind immer binär, man kann das aber leicht durch die zusätzliche Angabe der Option `-a` ändern. Zum Versenden via Email eignet sich besonders gut der Befehl `gpg -clearsign test.txt`, der die signierte Datei `test.txt.asc` erzeugt, deren Inhalt

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Need more herrings. Tux
```

```
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.2.1 (GNU/Linux)

iD8DBQE+cxKE/Puclrw5zucRAquDAJ0Uk8CVTRlOeJejzjF2X9p/1nYkJwCaAouI
pybyhiPqqCSVDfKDn5jQdSI=
=GCgO
-----END PGP SIGNATURE-----
```

dann leicht via Email verschickt werden kann. Überprüft wird mit `gpg -verify test.txt.asc`. (Hier ist `Need more herrings. Tux` der Inhalt der unsignierten Datei `test.txt`.)

Natürlich gibt es auch graphische Programme für GnuPG/PGP, und es ist auch möglich, GnuPG/PGP direkt in gängige Emailclients zu integrieren. Mehr dazu finden Sie z.B. auf der Homepage von GnuPG.

Am Ende möchte ich noch etwas zur Schlüsselverwaltung unter GnuPG sagen. Alle öffentlichen Schlüssel werden auf Ihrem öffentlichen Schlüsselbund gespeichert. Wie immer ergibt sich dabei das Problem der Echtheit der Schlüssel. Falls Sie den Schlüssel von einer Person erhalten haben, die Sie persönlich kennen, so ist es leicht, mit der Person zu sprechen und auf diesem Weg den Fingerabdruck des Schlüssels zu überprüfen. Sind Sie von der Echtheit eines Schlüssels überzeugt, so können Sie das GnuPG mitteilen: mit `gpg -edit-key` *keyid* kommen Sie in ein Menü, in dem Sie den Schlüssel mit der KeyID *keyid* bearbeiten können; alle zur Verfügung stehenden Befehle erhalten Sie, wenn Sie im Menü nach dem Prompt `help` eingeben. Zum Beispiel können Sie den Schlüssel mit `sign` signieren. Durch Ihre Unterschrift stufen Sie die Echtheit des Schlüssels ein. (Sie können in vier Abstufungen angeben, wie genau Sie diese Echtheit überprüft haben.)

Falls Sie diese Signatur auch an andere weitergeben (mit `gpg -export` *keyid*), so könnte jeder, der Ihnen vertraut, auch allen von Ihnen unterschriebenen Schlüsseln vertrauen. Das genau ist die Idee des **Web of Trust**.

Allerdings gibt es da noch ein kleines Problem: Auch wenn Sie die Echtheit der Unterschrift einer Person nicht anzweifeln (d.h., dass es sich wirklich um die Signatur der betreffenden Person handelt), so heißt dass noch lange nicht, dass Sie diese Person auch für zuverlässig halten. Es könnte ja Personen geben, die jeden Schlüssel unterschreiben, wenn man sie dafür nur auf ein Bier einlädt. Deshalb muss gesondert noch der Grad an Vertrauen in die Zuverlässigkeit einer Signatur mit dem Schlüssel festgelegt werden (`gpg -edit-key` *keyid* und dann den Befehl `trust` verwenden).

Um den Schlüssel- und Signaturaustausch zu vereinfachen, gibt es ein weltweites Netz aus **Keyservern**, die diese Informationen (öffentliche Schlüssel und zugehörige Signaturen) speichern. Alle Keyserver tauschen diese Informationen untereinander aus und somit reicht es, die Daten an einen einzigen Keyserver zu schicken. Das geschieht bei GnuPG, indem man anstelle der Option `-export` die Optionen `gpg -keyserver` *keyserver* `-send-key` *keyid* verwendet. Kennt man

die KeyID, so kann man den zugehörigen Schlüssel auch leicht mit `gpg -keyserver keyserver -recv-key keyid` importieren. Achtung: Ein auf diese Art importierter Schlüssel ist (falls er nicht von jemanden signiert ist, dem Sie vertrauen) vom Sicherheitsstandpunkt natürlich wertlos, solange Sie die Echtheit nicht über eine unabhängige Quelle überprüft haben (z.B.: offizielle Webseite).

Einer dieser Keyserver ist zum Beispiel `https://pgp.mit.edu/`. Dort gibt es auch ein Webformular, in dem man nach Schlüsseln suchen kann. (Falls Sie nach einer KeyID suchen, muss sie durch Voranstellen von `0x` als Hexadezimalzahl gekennzeichnet werden, also z.B.: `0x7DCBBBD6`.)

Ist ein Schlüssel einmal auf einem Keyserver, so ist es nicht möglich, diesen wieder zu löschen. Sie können ihren Schlüssel aber als ungültig erklären. Das geschieht mit einem sogenannten **Revocation Certificate**, das Sie mit dem Befehl `gpg -gen-rev keyid` erzeugen können (dazu benötigen Sie Ihren privaten Schlüssel, da die Echtheit durch Ihre Signatur bestätigt wird).

Am besten erstellt man bei der Schlüsselerzeugung auch gleich ein Revocation Certificate und verwahrt es an einem sicheren Platz. Falls der private Schlüssel verloren geht (oder man die Passphrase vergessen hat;-), kann man so auch ohne Kenntnis des privaten Schlüssels den Schlüssel jederzeit für ungültig erklären. Achtung: Ein Revocation Certificate ist nichts anderes als eine signierte Ungültigkeitserklärung für Ihren Schlüssel. Sie können damit weder den privaten Schlüssel wiederherstellen noch Dokumente entschlüsseln.

6.4 Kontrollfragen

Fragen zu Abschnitt 6.1: Digitale Signatur und Authentifizierung

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Digitale Signatur, Authentifizierung, DSA/DSS.

1. Richtig oder falsch?
   Wenn ich ein Dokument digital signiere, dann kann der Empfänger des Dokuments sichergehen, dass
   a) das Dokument von mir stammt
   b) das Dokument unverändert bei ihm angekommen ist.

   (Lösung zu Kontrollfrage 1)

2. Warum verwendet man für eine digitale Signatur zusätzlich zum asymmetrischen Verfahren auch eine kryptographische Hashfunktion?

   (Lösung zu Kontrollfrage 2)

3. Wie können Sie ein Dokument mit dem RSA-Algorithmus und SHA1 signieren?

   (Lösung zu Kontrollfrage 3)

4. Wie überprüfen Sie die Signatur eines mit RSA und SHA1 signierten Dokuments?

   (Lösung zu Kontrollfrage 4)

5. Wie können Sie Alice bitten, sich mit RSA zu authentifizieren?

   (Lösung zu Kontrollfrage 5)

6. Was tun Sie, wenn jemand Ihnen eine RSA-verschlüsselte Nachricht schickt, mit der Bitte, dass Sie sich authentifizieren?

   (Lösung zu Kontrollfrage 6)

7. Was ist der DSA? Was ist DSS?

   (Lösung zu Kontrollfrage 7)

Fragen zu Abschnitt 6.2: Zertifizierung

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Man-in-the-Middle-Angriff, Zertifizierungsstelle, Web of Trust.

1. Was versteht man unter dem Man-in-the-Middle-Angriff?

   (Lösung zu Kontrollfrage 1)

2. Wie verhindert man in der Praxis einen Man-in-the-Middle-Angriff?

   (Lösung zu Kontrollfrage 2)

3. Was ist die Aufgabe einer CA?

   (Lösung zu Kontrollfrage 3)

Fragen zu Abschnitt 6.3: PGP

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: PGP, GnuPG, Keyserver, Revocation Certificate.

1. Was ist der Unterschied zwischen PGP und GPG?

   (Lösung zu Kontrollfrage 1)

2. Was versteht man unter Web of Trust?

   (Lösung zu Kontrollfrage 2)

3. Kann PGP auch zum Signieren von Software verwendet werden?

   (Lösung zu Kontrollfrage 3)

Lösungen zu den Kontrollfragen

Lösungen zu Abschnitt 6.1

1. a) richtig b) richtig

2. Damit die Datenmenge nicht zu groß wird. Anstelle das gesamte Dokument zu verschlüsseln (signieren), wird nur der Hashwert des Dokumentes signiert (der das Dokument eindeutig charakterisiert).

3. Indem ich den Hashwert des Dokuments bilde, und dann den Hashwert mit meinem privaten Schlüssel verschlüssle. Dokument und Hashwert gebe ich gemeinsam bekannt.

4. Ich wende auf die Signatur den öffentlichen Schlüssel des Senders an, und vergleiche den erhaltenen Wert mit dem Hashwert des Dokuments.

5. Ich verschlüssle eine Zufallszahl mit dem öffentlichen Schlüssel von Alice und schicke diese an Alice. Wenn Alice die verschlüsselte Zahl entschlüsseln kann, so habe ich die Gewißheit, dass ich mit Alice kommuniziere.

6. Ich wende zuerst meinen geheimen Schlüssel und dann eine kryptographische Hashfunktion an. Ansonsten besteht die Gefahr, dass ein Mallory mir nicht einen Zufallstext, sondern den Hashwert eines für mich ungünstigen Dokuments schickt. Wenn ich darauf meinen privaten Schlüssel anwende, und das Ergebnis an Mallory zurückschicke, so hat er Dokument plus meiner gültigen Signatur.

7. DSA (= *Digital Signature Algorithm*) ist der Elgamal-Signaturalgorithmus gemeinsam mit SHA1. DSS ist der US-amerikanische Signaturstandard (= *Digital Signature Standard*), der heute neben DSA auch ECDSA sowie den RSA-SignaturAlgorithmus enthält.

Lösungen zu Abschnitt 6.2

1. Der Angreifer setzt sich in die Mitte der Verbindung und gibt beiden Teilnehmern vor, der jeweils andere zu sein.

2. Überprüfen des Schlüssels (in der Praxis dessen Fingerabdruck) über einen unabhängigen Kanal; oder Zertifizierung.

3. Eine CA bestätigt durch ihre digitale Signatur durch ein Zertifikat die Echtheit von öffentlichen Schlüsseln.

Lösungen zu Abschnitt 6.3

1. Beide implementieren den OpenPGP-Standard. PGP ist kommerzielle Software und nur für private Zwecke kostenlos nutzbar. GnuPG ist Open-Source und kann auch kommerziell kostenlos eingesetzt werden.

2. Dezentrales Zertifizierungsmodell öffentlicher PGP-Schlüssel.

3. Ja. Am besten mittels einer „detached signature“.

6.5 Übungen

Aufwärmübungen

1. RSA-Signatur mit Hashfunktion $H(x)$ = Buchstabensumme modulo 128 (z.B. $H(AA) = 65 + 65 = 2$):
   a) Ihr geheimer RSA-Schlüssel ist $(n, d) = (143, 103)$. Signieren Sie die Nachricht $x =$ „FAD“.
   b) Jemand sendet Ihnen die Nachricht $x =$ „LINUX“ zusammen mit der Signatur $s = 25$. Der öffentliche RSA-Schlüssel des Senders ist $(n, e) = (209, 53)$. Ist die Signatur gültig?
   (Umwandlung von Buchstaben in Zahlen gemäss ASCII-Code).

2. Ihr geheimer RSA-Schlüssel ist $(n, d) = (209, 17)$, Ihr öffentlicher Schlüssel ist $(n, e) = (209, 53)$. Die verwendete Hashfunktion H ist die Buchstabensumme modulo 128. Bob schickt Ihnen die Nachricht $r = (1, 162, 203)$ und bittet Sie, sich zu authentifizieren. Was antworten Sie? Was macht Bob mit Ihrer Antwort?

3. Ihr geheimer Elgamal-Schlüssel sei $(p, g, a) = (131, 6, 21)$, Ihr öffentlicher Schlüssel ist $(p, g, \alpha) = (131, 6, 110)$. Die verwendete Hashfunktion sei die Buchstabensumme modulo 128.
   a) Signieren Sie die Nachricht „FAD“.
   b) Wie überprüft Bob, ob die Signatur echt ist?

Weiterführende Aufgaben

1. **Signatur mit RSA** und Hashfunktion $H(x) = x \bmod 128$:
   a) Wie erzeugt Alice zu $p = 97, q = 31$ und $e = 17$ einen geheimen RSA-Schlüssel?
   b) Wie signiert Alice das Dokument $x = 1234$? Was schickt Alice an Bob?
   c) Bob erhält von Alice das signierte Dokument $(x, s) = (1234, 1786)$. Wie prüft Bob die Gültigkeit der Signatur?

d) Bob erhält bei anderer Gelegenheit von Alice die Dokumente $(x_1, s_1) = (210, 1786)$ und $(x_2, s_2) = (815, 2093)$. Sind die Signaturen gültig?

2. **Signatur mit Elgamal** und Hashfunktion $H(x) = x \bmod 128$:
   Alice wählt als Domain-Parameter $p = 29$ und $g = 2$ (Generator von $\mathbb{Z}_{29}^*$) und als geheimen Schlüssel $a = 7$. Sie gibt das zugehörige $\alpha = 12$ öffentlich bekannt.
   a) Nehmen Sie an, dass Alice als temporären Schlüssel $r = 3$ wählt. Wie signiert Alice das Dokument $x = 1234$? Was schickt Alice an Bob?
   b) Wie prüft Bob die Gültigkeit der Signatur?

3. Die Systemparamter für die Schnorr-Signatur seien $(p, g, m) = (167, 4, 83)$ und die Hashfunktion sei die Buchstabensumme modulo 128. Ihr geheimer Schnorr-Schlüssel sei $a = 33$, Ihr öffentlicher Schlüssel ist $\alpha = 29$.
   a) Signieren Sie die Nachricht „FAD“.
   b) Wie überprüft Bob, ob die Signatur echt ist?

4. Die Systemparamter für Elgamal seien $(p, g) = (167, 4)$ und die Hashfunktion sei die Buchstabensumme modulo 128. Homer hat den öffentlichen Schlüssel $\alpha = 62$ und die Nachricht „Nuclear“ mit $(54, 110)$ und die „Doughnut“ mit $(54, 78)$ signiert. Was hat Homer falsch gemacht? Wie lautet sein geheimer Schlüssel? Ist er eindeutig?

5. **Angriff auf eine Elgamal-Signatur**:
   Die Domain-Parameter $p = 29$ und $g = 2$ (Generator von $\mathbb{Z}_{29}^*$) und die „kryptographische“ Hashfunktion für das Signaturprotokoll sei wieder die einfache Hashfunktion $H(x) = x \bmod 128$.

   Nehmen Sie an, dass Alice den öffentlichen Schlüssel $\alpha = 12$ hat und leider zweimal als Nonce $r = 3$ wählt und damit die Dokumente $x_1 = 1234$ und $x_2 = 327$ signiert. Mallory fängt die Dokumente und Signaturen ab: $(x_1 = 1234, \rho = 8, s_1 = 18)$ und $(x_2 = 327, \rho = 8, s_2 = 5)$. Wie kann Mallory nun durch Lösen des Gleichungssystems

$$\begin{aligned} s_1 &= r^{-1}(h_1 - a\rho) \bmod p - 1 \\ s_2 &= r^{-1}(h_2 - a\rho) \bmod p - 1 \end{aligned}$$

   an r und den geheimen Schlüssel a von Alice gelangen? (Hinweis: die Gleichung für a hat hier keine eindeutige Lösung, sondern 4 mögliche Lösungen (siehe SB1, Satz 1.18). Mallory findet das richtige a indem er testet, ob $g^a = \alpha$.)

6. a) Finden Sie heraus, welche CAs Ihr Webbrowser akzeptiert.
   b) Welche CA hat das Zertifikat des Webservers der Uni Wien unterschrieben? Wann läuft es ab?

7. a) Finden Sie heraus welche elliptischen Kurven openssl unterstützt.
   b) Erzeugen Sie einen privaten Schlüssel (aus Kompatibilitätsgründen mit secp256k1 oder secp384r1).
   c) Erzeugen Sie einen Certificate Signing Request (CSR).
   d) Welche Möglichkeiten gibt es ein Gratis-Zertifikat zu bekommen? Berichten Sie über Ihre Erfahrungen.
   d) (optional) Führen Sie mit openssl den privaten Schlüssel und das Zertifikat zusammen und importieren sie diese in ihren Email-Client.

8. Erzeugen Sie sich mit gpg einen öffentlichen und privaten Schlüssel und
   a) exportieren Sie Ihren öffentlichen Schlüssel im ASCII-Format
   b) verschlüsseln Sie eine Datei
   c) entschlüsseln Sie diese Datei wieder.

9. Erzeugen Sie ein Revocation Certificate für Ihren Schlüssel.

10. Erzeugen Sie mit gpg eine signierte Textnachricht
    a) im default gpg-Format
    b) klartextsigniert (zum Versenden als Email)
    c) mit extra Signaturfile (detached signature).
    Überprüfen Sie diese Signaturen.

11. a) Finden Sie mithilfe eines keyservers heraus, wem der Schlüssel mit der KeyID 0x6708B463 gehört.
    b) Importieren Sie den Schlüssel.
    c) Signieren Sie ihn (Menü mit `gpg -edit-key` *keyid* und dann weiter mit `sign` bzw. ggf. `help`; Antwort mit „yes“ oder „no“ bzw. wie vorgeschlagen).

Lösungen zu den Aufwärmübungen

1. a) Die Nachricht im ASCII-Code lautet $x = (70, 65, 68)$ und der Hashwert ist $h = H(x) = 75$. Verschlüsselung mit geheimem Schlüssel ergibt die Signatur $s = 75^{103} = 36 \,(\mathrm{mod}\, 143)$. Signierte Nachricht ist $(\mathrm{FAD}, 36)$.
   b) Die Nachricht im ASCII-Code lautet $x = (76, 73, 78, 85, 88)$, ihr Hashwert ist $h = H(x) = 16 = v_2$. Öffentlicher Schlüssel auf Signatur angewendet: $v_1 = 25^{53} = 16 (\mathrm{mod}\, 209)$. Wegen $v_1 = v_2$ ist die Nachricht gültig.

2. Sie entschlüsseln mit Ihrem geheimen Schlüssel und erhalten die Nachricht $x = (1, 2, 3)$. Die Antwort an Bob ist der Hashwert $H(x) = 6$. Bob berechnet den Hashwert seiner ursprünglichen Nachricht und vergleicht ihn mit Ihrer Antwort.

3. a) Der Hashwert der Nachricht ist $h = 75$. Wähle z.B. $r = 37$ und berechne $\rho = 93$ und $r^{-1} = 123 \,(\mathrm{mod}\, 130)$. Damit ergibt sich $s = 123(75 - 93 \cdot 21) = 16 \,(\mathrm{mod}\, 130)$. Signierte Nachricht ist $(\mathrm{FAD}, 93, 16)$.

b) Bob berechnet $v_1 = 110^{93} \cdot 93^{16} = 18 \pmod{131}$ und $v_2 = 6^{75} = 18 \pmod{131}$. Da beide Werte übereinstimmen, ist die Signatur echt.

Lösungen zu ausgewählten Aufgaben

1. a) $n = 3007$, $d = 2033$
   b) Alice sendet $(x, s) = (1234, 1786)$
   c) ...
   d) erstes Dokument: Signatur gültig; zweites Dokument: Signatur nicht gültig.

2. a) $\alpha = 12, \rho = 8, s = 18, x = 1234$
   b) $v_1 = v_2 = 22$

3. Mit (z.B.) $r = 42$ folgt $(s, h) = (17, 42)$.

4. Da $\rho_1 = \rho_2$ hat er vergessen die Nonce zu ändern! $a = 12$ oder $a = 95$

5. Gleichungssystem nach den beiden Unbekannten r und a lösen: $r = 3$, $a = 7$

6. –

7. a) `openssl ecparam -list_curves`
   b) `openssl ecparam -name secp384r1 -genkey -noout -out private-key.pem`
   c) `openssl req -new -key private-key.pem -out cert.csr`
   d) –
   e) `openssl pkcs12 -export -out certs.p12 -inkey private-key.pem -in certs.pem`

8. `gpg -gen-key`
   a) `gpg -export -a` *`keyid`* `>` *`mykey.txt`*
   b) `gpg -r` *`keyid`* `-e` *`filename`*
   c) `gpg` *`filename`*`.gpg`

9. `gpg -o` *`rev.txt`* `-gen-rev` *`keyid`*
   Die Option -o rev.txt bewirkt die Ausgabe in ein File rev.txt. Das Revocation Certificate wird bei Bedarf an den keyserver bzw. an alle Leute, die davon wissen sollten, geschickt.

10. a) signiere mit `gpg -s file` bzw. überprüfe mit `gpg file.gpg`
    b) `gpg -clearsign file` bzw. `gpg file.asc`
    c) `gpg -b file` bzw. `gpg -verify file.sig file`

11. a) Der Schlüssel gehört Susanne Teschl.
    b) `gpg -keyserver wwwkeys.pgp.net -recv-key 6708B463`
    c) `gpg -edit-key 6708B463`
    (Wenn Sie zu einem Schlüssel, der auf einem keyserver liegt, Ihre Signatur

hinzufügen möchten - für den angegebenen Schlüssel allerdings nicht notwendig :-) - dann: Schlüssel importieren - signieren - exportieren. Beim Exportieren eines Schlüssels auf einen keyserver werden automatisch alle Signaturen mitexportiert.

Literatur

[FIPS186-5] NIST. „Digital Signature Standard (DSS)“. In: *Federal Information Processing Standards Publication (FIPS)* 186-5 (2023). URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf.

[RFC6979] T. Pornin. „Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)“. In: *Request for Comments* 6979 (2013). URL: https://www.rfc-editor.org/rfc/rfc6979.

[RFC8017] K. M. Moriarty u. a. „PKCS #1: RSA Cryptography Specifications“. In: *Request for Comments* 8017 (2016). URL: https://www.rfc-editor.org/rfc/rfc8017.

[RS84] R. L. Rivest und A. Shamir. „How to Expose an Eavesdropper“. In: *Communications of the ACM* 27.4 (1984), S. 393–395. URL: https://doi.org/10.1145/358027.358053.