Seminarvortrag von Evgeniya Zaytseva

Seminar:	**Kryptographie**
Semester:	**WS 05/06**
Dozent:	**Prof. Dr. Martin Kreuzer**
Homepage:	http://www.matha.mathematik.uni-dortmund.de/~kreuzer/krysem.html
Thema:	**E-Mailsicherheit und PGP**
Universität:	**Dortmund**

Pretty Good Privacy – PGP

Inhaltsangabe Seite

Pretty Good Privacy PGP-Einführung

Programm	als Freeware-Version (und als kommerzielle Version)
Zweck:	-Datenverschlüsselung bei E-Mails und bei der Speicherung von Daten -Authentifizierung -Integrität
PGP:	stellt nur die kryptographischen Dienste zur Verfügung →der eigentliche Versand erfolgt über das Netzwerk über Standardprogramme →die im Internet vorhandenen Mail-Infrastrukturen von PGP genutzt funktioniert unabhängig vom Betriebssystem oder Prozessoren →auf jedem Computer nutzbar Verrichtung der Verschlüsselungsarbeit nicht unsichtbar im Hintergrund →Verschlüsselungsprozess vom Nutzer angestoßen Interaktion mit dem Nutzer durch Eingabe eines Passwortes →für Gewährleistung der Sicherheit der Signatur und der Entschlüsselung entscheidender Schritt!
E-Mail-PlugIns:	Freeware-Version enthält standardmäßig Plug-Ins für die E-Mail-Programme Outlook, Outlook Express, Eudora,…
Beispiel:	Installation des Plug-Ins→Outlook erhält zwei weitere Buttons (Decrypt/ Verify Message und Launch PGP Keys) und ein „PGP“-Menue mögliche Einstellungen: • ob Nachrichten immer verschlüsselt und/oder signiert werden sollen • ob Standard PGP verwendet werden soll • ob Nachrichten beim Öffnen automatisch entschlüsselt und verifiziert werden sollen • ob eine entschlüsselte Nachricht nur mit einem sicheren Anzeigeprogramm dargestellt werden soll

Geschichte und Hintergründe

PGP: im Wesentlichen nur von Phil Zimmermann entworfen und verbreitet

- Computerspezialist aus Boulder, Colorado(USA)
- studierte in den 70er Jahren an der Florida Atlantic University Physik und Computerwissenschaften

1991: Erscheinen einer Gesetzesvorlage (USA), dass jede Verschlüsselungssoftware eine Hintertür für den staatlichen Zugriff enthalten müsse

→Zimmermanns Ziele:

- RSA-Verschlüsselungssoftware für die breite Öffentlichkeit, für Privatleute entwickeln
- wirtschaftlich und schnell
- passend für die Leistungsfähigkeit eines durchschnittlichen Computers
- anwenderfreundliche Programmoberfläche

Probleme:

1. RSA war patentiertes Produkt→Lizenz von RSA Data Security nötig
2. USA Exportbestimmung: Klassifizierung von Krypto-Software als Waffe→Waffen aber mit Exportverbot belegt

5 Juni 1991: Phil Zimmermann verteilt PGP 1.0 über einen Freund im Internet

- symmetr. Alg.: Bass-O-Matic →selbstentworfen
- Public-Key-Operationen: RSA
- Hashwertbildung: MD4
- Kompressionsalg.: LZ Huf
- 7-Bit-Transportcodierung: uuencode

In der Dokumentation fordert Phil Zimmermann die Nutzer auf, sich vor der Nutzung eine Lizenz zu besorgen.
RSA Data SEcurity Inc fordert Phil Zimmermann auf, die Verteilung von PGP einzustellen wegen fehlender Lizenz. Er ignoriert es.

2 Sept. 1992: PGP 2.0 außerhalb von USA veröffentlicht:
erste Version, wie wir sie heute kennen, entwickelt zusammen mit einem Team aus Neuseeland und Europa

- IDEA
- RSA
- MD5
- ZIP-Kompression
- BASE 64 →Umwandlung des Binärfiles in ein Textfile

→schnell große Fangemeinde auf der ganzen Welt!

Anklage: 14Sept 1993:
Büro des US-Zolls (San Jose, Kalifornien) erhebt Anklage gegen Phil Zimmermann→Zimmermann als Waffenhändler

…zahlreiche Veröffentlichungen von unterschiedlichen PGP-Versionen folgen
…USA, Europa getrennt zu betrachten (Patentfrage) …
Anpassungen
Verbesserungen
Ausbau des Programms

August 1997: Um die Konsistenz zwischen der US- und der internationalen Version zu gewährleisten, nutzen die Entwickler das Recht auf Pressefreiheit der USA, (denn Exportverbot von Büchern würde Meinungsfreiheit untersagen)
→Kompletter Source Code von PGP wird in Buchform publiziert mit Seitenzahlen als C-Kommentare („/* */“).
→Neue Features von neuen Versionen konnten einfach eingescannt werden.
Nov 1998: Veröffentlichung eines Standard-Dokuments OpenPGP

Symmetrische Kryptographie vs. Asymmetrische Kryptographie

Symmetrische Kryptographie

- Sender und Empfänger besitzen einen gemeinsamen Geheimschlüssel
- können beide unter dessen Verwendung auf einen Verschlüsselungs-/ Entschlüsselungsalgorithmus zugreifen
- bei N Teilnehmern: muss jeder N-1 Schlüssel geheim halten

+ effizient implementierbar

- Sender und Empfänger müssen einen gemeinsamen Schlüssel vereinbaren
→keine spontane Kommunikation möglich

- Sender und Empfänger müssen sich bzgl. Schlüsselgeheimhaltung vertrauen
→hoher Schlüsselverwaltungsaufwand

Asymmetrische Kryptographie Public-Key-Kryptographie

- jeder Teilnehmer T hat zwei Schlüssel: öffentlichen Schlüssel E_T
  privaten Schlüssel D_T

- Ver-/ und Entschlüsselung durch Einwegfunktion mit Falltür gegeben
  A will B eine Nachricht senden:
  1. A sucht B’s öffentlichen Schlüssel raus
  2. A wendet diesen auf die zu sendende Nachricht an
  3. A sendet den Geheimtext an B
  4. nur B kann den Text mit seinem privaten Schlüssel entschlüsseln

\+ spontane Kommunikation zwischen Sender und Empfänger jederzeit möglich
- nur wenige Verfahren bekannt
- alle mathematisch ähnlich
- alle bis heute bekannten Verfahren wegen Arithmetik großer Zahlen sehr langsam

→ in der Praxis: **Hybridverfahren!**

Kombination aus symmetrischer und asymmetrischer Kryptographie und der jeweiligen Vorteile!

Vorteile: mit + gekennzeichnet
Nachteile: mit - gekennzeichnet

Hybrides Verfahren bei PGP

1. Sender einer Nachricht wählt eine Zufallszahl als symmetrischen Schlüssel
2. Mit diesem Schlüssel und einem symmetrischen Verschlüsselungsverfahren wird die Nachricht verschlüsselt.
   (Als Betriebsmodus wird der Cipher-Feedback-Modus verwendet.)
   (Algorithmus: DES, Triple-DES, CAST oder IDEA)
3. Danach verschlüsselt der Sender die Zufallszahl mit dem öffentlichen RSA- oder ElGamal-Schlüssel des Empfängers.

Kombination der Vorteile:

Symmetr. Kr.: →deutlich effizientere Durchführung bei langen Nachrichten
asymmetr. Kr.: →Lösung des Problems der Schlüsselvereinbarung durch asymmetrische Verschlüsselung des symmetrischen Schlüssels

weiterer Vorteil: jede Nachricht besitzt ihren eigenen symmetrischen Schlüssel

→Angreifer erhält durch Belauschen der Kommunikation keine großen Mengen an Geheimtexten, die unter demselben Schlüssel chiffriert worden sind

Asymmetrisches Verfahren:

verwendet immer denselben öffentlichen Schlüssel, aber es werden nur kleine Zufallszahlen verschlüsselt
→Erschwerung der Kryptoanalyse des Public-Key-Verschlüsselungsverfahrens ebenfalls

Authentizität der Daten:

gewährleistet durch digitale Signaturen

1. erst eine Hashfunktion (SHA-1 oder MD5) auf eine Nachricht anwenden→Sicherung der Integrität
2. Hashwert mit dem RSA- oder ElGamal- Verfahren (DSS) mit dem privaten Schlüssel des Empfängers signieren
3. Anhängen einer Zeitvariable über den Zeitpunkt der Signaturerstellung

Kombination der beiden Verfahren bei PGP:

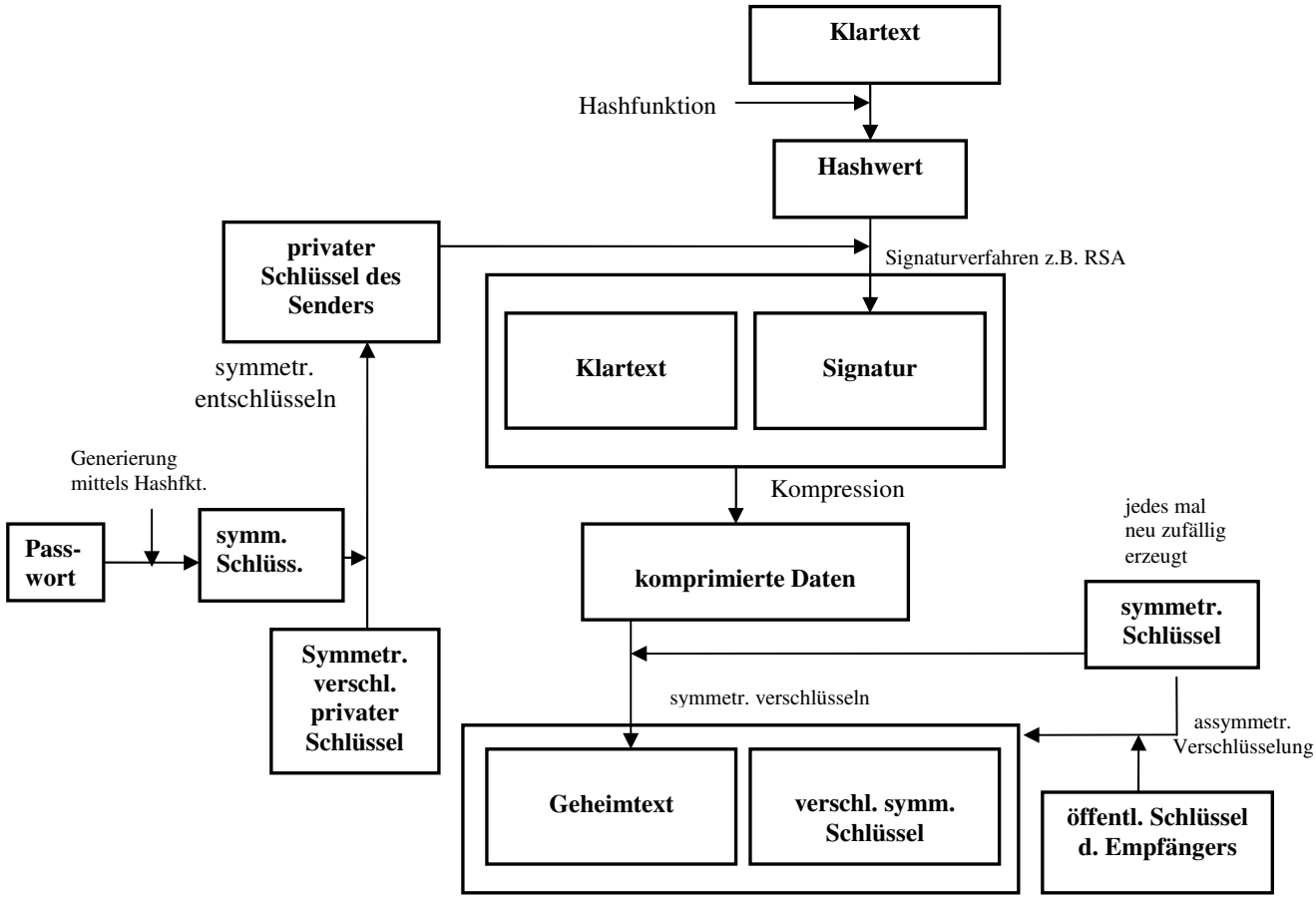
Sender signiert die Nachricht erst und verschlüsselt sie dann zusammen mit der Signatur.

Verwendung von Datenkompressionsverfahren (PKZip)

→für eine bessere Effizienz
angewendet nach der Signaturerstellung und vor der Verschlüsselung
→Vorteil: Für die Verifikation der Signatur werden auch die unkomprimierten Daten benötigt, die der Empfänger im Allgemeinen speichert
sonst: müsste der Empfänger auch die komprimierten Daten speichern, da das von PGP verwendete Kompressionsverfahren nicht deterministisch ist, d.h. derselbe Datensatz kann in zwei Durchgängen unterschiedlich komprimiert werden

Kompression vor der Verschlüsselung:
→Vorteil: Daten weisen danach eine geringere Redundanz auf
→Kryptoanalyse von verschlüsselten komprimierten Daten schwieriger als von verschlüsselten unkomprimierten Daten

Struktur des Verschlüsselungsablaufs eines Klartextes



(Generierung der Schlüssel wird in Kapitel Schlüsselverwaltung näher erläutert.)

Hashfunktionen

Allgemeine Funktionsweise von Hashfunktionen

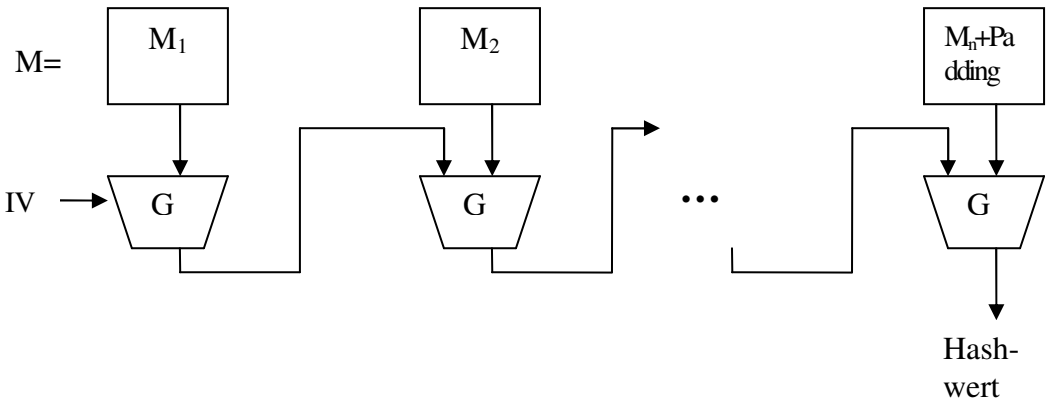
Hashfunktion: nicht injektive Abbildung H: Universum → Adressbereich
Element ↦ Hashadresse des Elements
speziell: H: A_1*→A_2*, A_1 ,A_2 endliche Alphabete
erzeugt zu einem beliebig langen Wort M aus A_1* einen Wert H(M), den Hashwert (message digest) fester Länge

Anforderungen an eine Hashfunktion: →Sicherung der Integrität von Daten

1. H ist Einweg-Funktion:
   Für jedes M∈ A_1* gibt es ein effizientes Verfahren zur Berechnung von H(M).
   Bei gegebenem h=H(M) gibt es kein effizientes Verfahren, um M zu berechnen.
2. H ist kollisionsfrei:
   Bei gegebenem h=H(M) für ein M∈ A_1* ist es (fast) unmöglich, eine Nachricht M’ ≠ M zu finden mit H(M’)=h, M’∈ A_1*.

Konstruktion von Hashfunktionen:

Hashfunktionene sind realisiert durch eine Folge gleichartiger Kompressionsfunktionen, durch welche die Eingabe M blockweise zu einem Hashwert verarbeitet wird.



IV: Initialisierungswert (festgelegt, Bestandteil der Spezifikation von Hash-Algorithmen)
G: verwendete Kompressionsfunktion der Hashfunktion H
f: A_1^k → A_2^k, f(0)=IV
f(i)=G(f(i-1),M_i) M= M_1,M_2,…, M_n i=1,…,n
H(M)=f(n)=h Hashwert von M

Klassen kryptographischer Hashfunktionen (abhängig vom Design von G)

1. Hashfunktionen basierend auf symmetrischen Blockchiffren
   → einfachere Verwendung
2. dedizierte Hashfunktionen z.B. SHA-1, MD5
   Kompressionsfunktionen speziell für die Erzeugung von Hashwerten konstruiert
   →deutlich effizientere Berechnung

Secure Hach Algorithm SHA-1

Nachricht M: als Bitfolge betrachtet
Nachrichtenlänge: Anzahl der Bits
(Falls die Nachrichtenlänge ein Vielfaches von 8 ist, ist M in Hexadezimal ausdrückbar, was hier zur Kompaktheit der Darstellung verwendet wird.)

1) Zerlegung der Nachricht in 512-Bit Blöcke $M_1,M_2,\ldots, M_n$

Falls die Länge des letzten Blocks < 512 ist wird ein Padding (Auffüllen des letzten Blocks) durchgeführt:

Padding: Annahme: Nachrichtenlänge $L < 2^{64}$

- Anhängen einer 1
- Anhängen von 0,…,0
- Die letzten 64 Bit sind reserviert für die Länge L der Originalnachricht

n-ter Block: **M_n100…0L :** insgesamt 512 Bit

Nachricht nach dem Padding: n Blöcke (zu je 512 Bit): $M_1,M_2,\ldots, M_n$
(M_1 enthält die ersten Bits der Nachricht)

2) Verwendete Funktionen

logische Funktionen $f_0, f_1,\ldots, f_{79}$
$f_t: \{0,1\}^{32} x \{0,1\}^{32} x \{0,1\}^{32} \rightarrow \{0,1\}^{32}$
$(B,C,D) \mapsto f_t(B,C,D)$
32-Bit Wörter

$f_t(B,C,D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad (0 <= t <= 19)$

$f_t(B,C,D) = B \text{ XOR } C \text{ XOR } D \quad (20 <= t <= 39)$

$f_t(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 <= t <= 59)$

$f_t(B,C,D) = B \text{ XOR } C \text{ XOR } D \quad (60 <= t <= 79)$

3) Verwendete Konstanten

K(0), K(1),…,K(79)

$K = 5A827999 \quad (0 <= t <= 19)$

$K_t = 6ED9EBA1 \quad (20 <= t <= 39)$

K_t = 8F1BBCDC (40 <= t <= 59)

K_t = CA62C1D6 (60 <= t <= 79)

4) Rechenoperationen

AND: bitweise logisches „und“
OR: bitweise logisches „oder“
XOR: bitweise logisches „exklusives oder“
NOT: bitweise logisches Komplement

Z=X+Y: X und Y werden als Integer-Zahlen x, y aufgefasst.
$0 \leq x,y<2^{32}$ $z=(x+y) \bmod 2^{32}$
z wird wieder zum Wort Z konvertiert.

$S^n(X)$: zyklischer Linsshift um n Stellen
$S^n(X)=(X<<n)OR(X>>32-n)$
X<<n: Entferne links n Stellen und fülle von rechts mit n Nullen auf
X>>32-n: Entferne rechts 32-n Stellen und fülle von links mit 32-n Nullen auf

5) Berechnung des Hashwertes

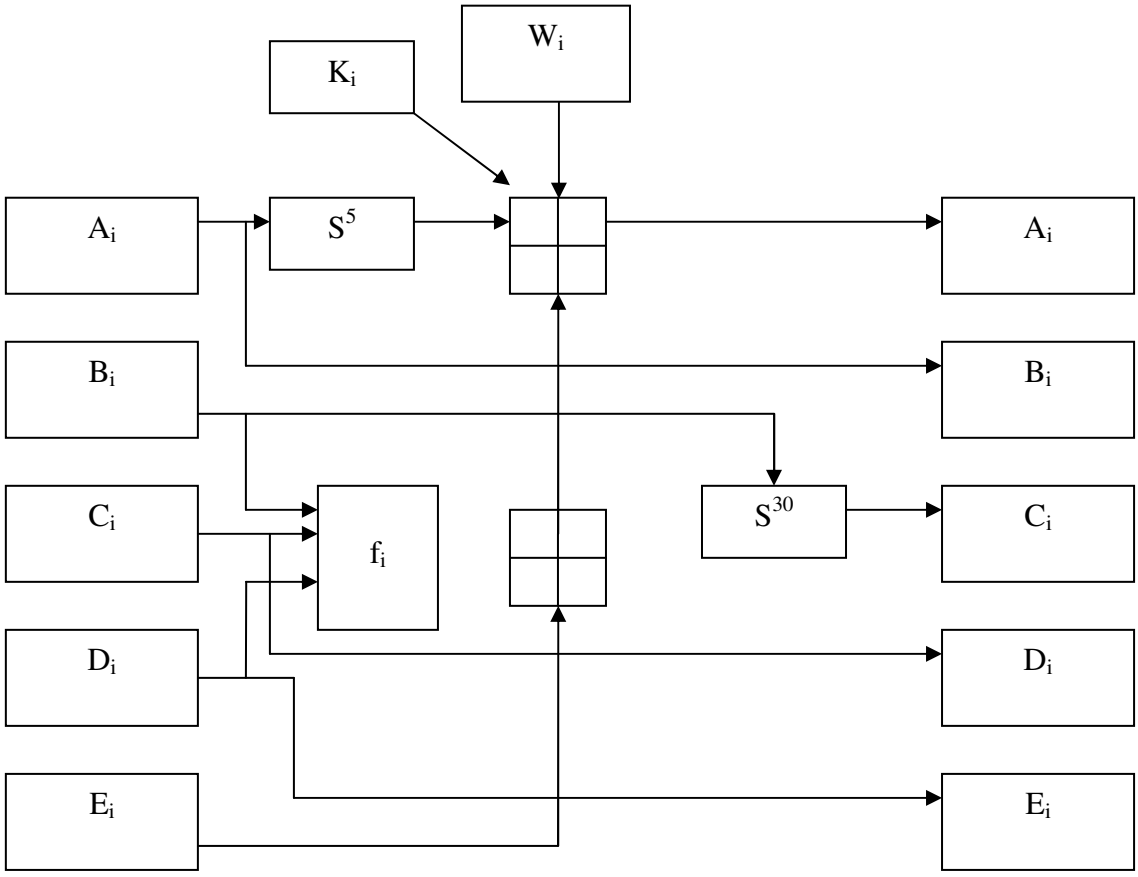
Verwendung: Nachrichtenblöcke (nach Padding) $M_1,M_2,\ldots, M_n$
1 Puffer: aus 5 32-Bit Wörtern A, B, C, D, E
2 Puffer: aus 5 32-Bit Wörtern H_0, H_1, H_2, H_3, H_4
80 Wörter: W_0, W_1,..., W_{79}

Generierung des Hashwertes

i) $M_1,M_2,\ldots, M_n$ werden sequentiell abgearbeitet, für jedes M_i 80 Schritte
ii) Initialisierung der H_i s (in hex): H_0 = 67452301
H_1 = EFCDAB89
H_2 = 98BADCFE
H_3 = 10325476
H_4 = C3D2E1F0
iii) Bearbeitung von M_i
1. Teile M_i in 16 Worte W_0, W_1,..., W_{79}
2. Die 16 Worte werden zu 80 Worten expandiert
Von t = 16 bis 79 setze $W_t = S^1(W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR $W_{t-16})$
3. Setze $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$
4. Von t = 0 bis 79 do
$TEMP = S^5(A) + f_t(B,C,D) + E + W_t + K$
$E = D; D = C; C = S^{30}(B); B = A; A = TEMP;$
5. Setze
$H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$

Nach n-facher Berechnung bis einschließlich M_n
erhält man den Hashwert: 160-Bit String H_0, H_1, H_2, H_3, H_4

Anwendung von Schritt 4 auf ein Wort W_i



Schlüsselverwaltung

Speicherung der Schlüssel: in zwei Dateien

1. pubring.pkr (Public Key Ring): fremde öffentliche Schlüssel der Kommunikationspartner
2. secring.skr (Private Key Ring): eigene Schlüsselpaare

Schlüsseltypen

1. symmetrischer Schlüssel (Sitzungsschlüssel) für den einmaligen Gebrauch
2. privater Schlüssel →secring.skr
3. öffentlicher Schlüssel→secring.skr
4. Passwort basierter symmetrischer Schlüssel

zu 1.: Erzeugung von einmal verwendbaren symmetr. Schlüsseln

- durch einen symmetr. Verschlüsselungsalg. (im CFB-Modus)
- als Initialisierung wird z.B. Verzögerungszeiten bei Tastatureingabe oder Mausbewegung des Benutzers vom Pseudozufallsgenerator PRNG verwendet

zu 2/3/4: Erzeugung eines privaten Schlüssels (und des passenden öffentlichen Schlüssels)

bei Neuinstallation von PGP:

- bei Neuinstallation von PGP: Übernahme von vorhandenen Schlüsselpaaren möglich
- Neuerzeugung mit „PGP Key Generation Wizard“: „New Key“ im Menue „Keys“ →dieser Assistent führt durch eine Standardprozedur zur Erzeugung eines Schlüsselpaares

Eigene Einstellungen unter „Expert“ möglich:

			Voreinstellungen
i.	Auswahl eines P-K-Algorithmus:	Diffie-Hellmann/DSS	DSS
		RSA	sehr
		RSA Legacy	sicher
ii.	Auswahl der Schlüssellänge von 1024 bis 4096 Bit		2048
iii.	Angabe der Gültigkeitsdauer des erzeugten Schlüssels		Begrenzung wichtig!

Die Angabe der Gültigkeitsdauer des privaten Schlüsselpaares ist sehr wichtig, da z.B. in zwei Jahren es zum Vergessen des Passwortes, Verlust der Datei oder neuen kryptografischen Angriffen kommen könnte und man sich unnötig mit veralterten Public Keys herumschlagen müsste.

Angabe der E-Mail-Adresse wichtig!

E-Mail-Plug-In wählt den öffentlichen Schlüssel zum Chiffrieren des Sitzungsschlüssels durch Suchen der E-Mail-Adresse des Empfängers

Erzeugung:

1. oben angegebene Vorauswahlen
2. Angabe der E-Mail-Adresse
3. Benutzer wird aufgefordert eine Passphrase einzugeben
   →wichtigster Schritt für die Sicherheit von PGP!
   PGP: -verlangt langen Text oder eine Zeichenfolge, die viele Sonderzeichen enthält als komplexes Passwort
   -warnt, wenn Passphrase zu einfach (Güte an einem Balken ablesbar)
4. mittels einer Hashfunktion (SHA-1) wird aus dem Passwort ein symmetrischer Schlüssel (4.) generiert
5. Passwort wird anschließend gelöscht
6. Verschlüsselung des privaten Schlüssels des Benutzers mit dem erzeugten symmetrischen Schlüssel
7. anschließend Löschen des symmetrischen Schlüssels

- falls der private Schlüssel zum Entschlüsseln oder Signieren gebraucht wird:
  →Benutzer gibt Passwort ein
  →symmetrischer Schlüssel zum Entschlüsseln des privaten Schlüssels wird generiert
- PGP speichert weder Passwort noch den symmetrischen Schlüssel
  →bei Vergessen des Passworts kann privater Schlüssel nicht mehr entschlüsselt werden, wird nutzlos

Private Key Ring:

für jeden öffentlichen Schlüssel des Benutzers:
Speicherung in verschlüsselter Form:

- des privaten Schlüssels
- einer Identifikationsnummer Key ID
- des Zeitpunktes der Schlüsselerzeugung
- einer Benutzeridentifikation (E-Mail-Adresse)

Public Key Ring:

Speicherung:

- des öffentlichen Schlüssels der Kommunikationspartner des Benutzers
- der Zeitangabe derer Erzeugung
- Identifikationsnummern: bei Signatur oder Verschlüsseln von Nachrichten stets angehängt

Übertragung des öffentlichen Schlüssels:

- PGP gibt kein festes Verfahren vor
- alle öffentlichen Schlüssel mit einem vorgegebenen Format werden akzeptiert
- Benutzer gibt selbst an, welches Vertrauen er einem bestimmten Schlüssel entgegenbringt

Vertauensmodelle:

- Direct Trust:

  Jeder Nutzer muss selbst entscheiden, welche öffentliche Schlüssel er als vertrauenswürdig ansieht. Das Vertrauen kann durch direkten Kontakt mit dem Eigentümer oder eine telefonische Verifikation des Hashwertes des Schlüssels erreicht werden.

- Web of Trust:

  Wenn A B vertraut, kann er auch allen von B signierten Schlüsseln vertrauen. So entsteht ein Netz von Vertrauensbeziehungen. Aber: Je größer das Netz wird, desteo größer ist auch die Gefahr, dass ein zu vertrauensseliges Mitglied dabei ist. Jeder kann die Tür für einen Angreifer öffnen.
  →Kann auch sehr gefährlich sein.

- hierarchische Vertrauensmodelle:

  Hier werden hohe Sicherheitsanforderungen an die Instanzen, die die öffentlichen Schlüssel signieren dürfen (Certification Authorities) gestellt. Angewendet wird dieses Modell vor allem in Firmen, weil diese auch hierarchisch strukturiert sind. Es gibt dann einen speziellen Schlüsseltyp zur Zertifizierung von öffentlichen Schlüsseln: den Certification Key.

- Cumulative Trust :

  Einem öffentlichem Schlüssel wird vertraut, falls eine bestimmte Anzahl von vertrauenswürdigen Personen diesem Schlüssel auch vertraut.

Open PGP – der Standard

beschreibt Struktur aller PGP Nachrichten und Prozeduren ihrer Generierung

Basis: Datenformate der Software-Version 5.0 PGP

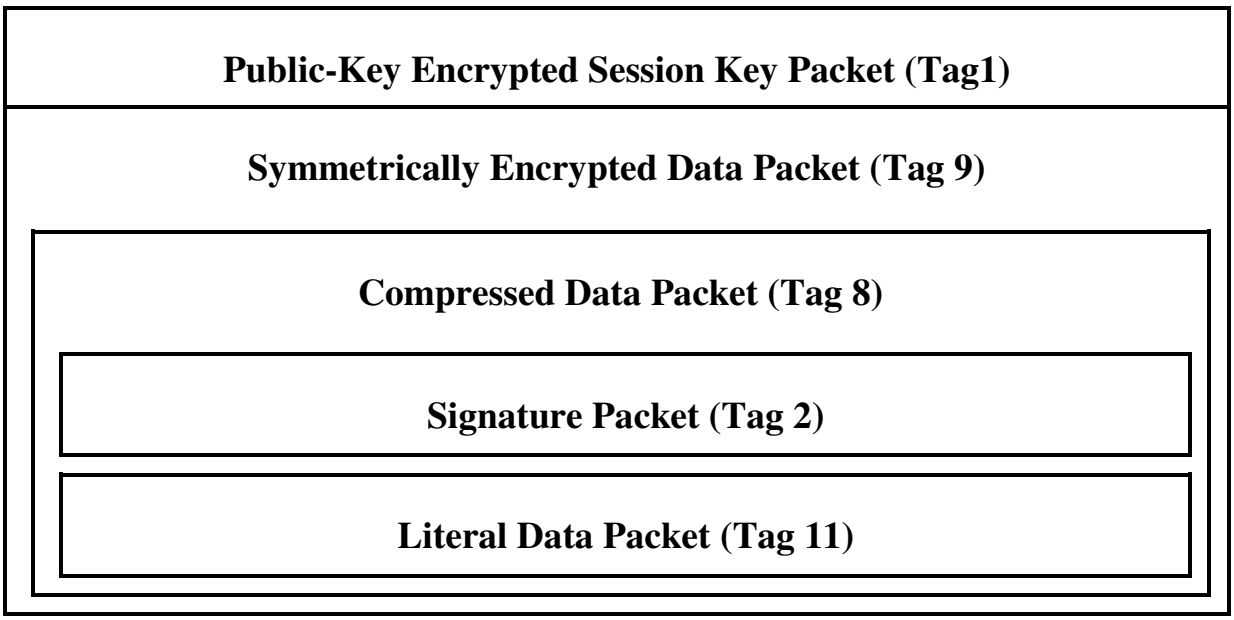
PGP-Nachrichten: aus PGP-Paketen (Datensätzen) zusammengesetzt

Struktur von PGP-Paketen:
- →Header:
  - →Tag: Angabe der Interpretation der nachfolgenden Daten
  - →Länge: 1-5 Bytes mit der codierten Länge des nachfolgenden Dateiteils codiert
- →Body: Inhalt abhängig vom Tag

wichtigste Pakete:

- Literal Data Packet (Tag11): die zu schützenden Daten
- Signature Packet (Tag2):
  - Tag-Angabe
  - Angabe der Version
  - Längenangabe der folgenden gehashten Bytes
    zuerst Literal- Data-Packet gehasht
    5 Byte „Type“ und „Creation Time“ gehasht
  - Angabe der Signer Key ID
  - 1 Byte P-K-Signaturfktion und Hashfktion
  - Signatur
- Compressed Data Packet (Tag8): Komprimierung des Gesamtpackets
- Symmetrically Encrypted Session Key Packet (Tag1):
  Verschlüsselung des Sitzungsschlüssels mit dem öffentlichen Schlüssel des Empfänger

Komplette OpenPGP-Nachricht



PGP – Die Angriffe

wichtiges Argument für die Sicherheit von PGP:

Der Sourcecode von PGP ist öffentlich bekannt und kann von jedem Nutzer überprüft werden.
→Entgegen der vermuteten Absicht von Geheimdiensten, versteckte „Hintertüren“ in Verschlüsselungssoftware einzubauen, die das Mitlesen von vertraulichen Nachrichten durch diese Geheimdienste ermöglichen sollten.

Aber übersehen:

unbeabsichtigte Sicherheitslücken durch fehlerhafte Spezifikation oder Implementierung von PGP

Angriff 1 auf den öffentlichen Schlüssel Additional Decryption Key

Version 5.5 von PGP:

Möglichkeit zum öffentlichen Schlüssel einen weiteren öffentlichen Schlüssel hinzu zubinden, den ADK

→als Feature: hinzugefügt, um Firmen die kontrollierte Entschlüsselung der E-Mails und Files ihrer Mitarbeiter zu ermöglichen

ADK: wird wie der öffentliche Schlüssel eines weiteren Empfängers betrachtet, an den die Nachricht gesandt bzw. für den das File verschlüsselt werden soll

neu dabei!: Der in der Firma eingesetzte PGP-Client verschlüsselt nur dann Daten, wenn er einen ADK findet.

Public-Key-Pakete: zwei Versionen:
beide enthalten die mathematischen Zahlenwerte für die verschiedenen public-Key-Algorithmen

alte Version 3: arbeitet nur mit RSA, enthält den Zeitpunkt der Erzeugung und Zusatzinfos zum Schlüssel (z.B. seine Gültigkeitsdauer)

neue Version 4: alle Zusatzinfos zum Schlüssel (außer dem Zeitpunkt seiner Erzeugung) sind in ein anderes Datenformat ausgelagert →in das Signatur-Datenformat der gleichen Version 4

Ziel der Umstellung:
beliebige Zusatzinfos an das Schlüssel binden zu können (z.B. bevorzugten symmetrischen Verschlüsselungsalgorithmus, JPEG-Bild des Benutzers) auch nach der Erzeugung des Schlüssels →nur durch Erzeugung einer Selbstsignatur

Datenstruktur eines Public-Key-Pakets:

neue Version 4: alle Zusatzinfos zum Schlüssel (außer Zeitpkt. seiner Erzeugung sind in ein anderes Datenformat ausgelagert
→in das Signatur-Datenformat

Ziel der Umstellung: beliebige Zusatzinfos an den Schlüssel binden zu können, auch nach seiner Erzeugung →nur durch Erzeugung einer Selbstsignatur

Einfügen der Zusatzinformation:

→als „Hashed Subpackets“ durch Signatur gegen Veränderung geschützt

→als „Non-Hashed Subpackets“, beliebig veränderbar ohne Beeinträchtigung der Gültigkeit der Signatur

bei ADK:

wird in der Version 4-Signatur eingefügt und dort als „required“ eingestuft statt der möglichen „welcome“-Einordnung

➔d.h. dieses Subpacket muss bei der Verwendung des Public Key und seiner Signatur unbedingt beachtet werden: es enthält die Schlüssel-ID eines fremden Schlüssels, mit dem der Sitzungsschlüssel ebenfalls verschlüsselt werden muss

Also: ADK fließt in den Hashwert ein, der signiert wird. Keine Veränderung möglich?

Idee: ein ADK-Subpacket als „Non- Hashed Subpacket“ in die Signatur integrieren

➔Sitzungsschlüssel könnte ein zweites Mal mit dem eingeschleusten ADK verschlüsselt werden

Angriffsszenario:

- Angreifer A liest verschiedene PGP Public Keys von einem PGP-Schlüsselserver, darunter auch den Schl. des Empfängers E
- A verändert die Schlüssel durch Einfügen des eigenen ADKs als NonHashed Subpacket
  Signatur des Schlüssels bleibt gültig
- A speichert die manipulierten Schlüssel wieder auf dem PGP-Schlüsselserver
- Sender S möchte verschl. Nachricht an Empfänger E senden, dessen E-Mail Adresse er kennt
  →sucht dessen öffentlichen Schlüssel auf dem Schlüsselserver
- S lädt manipulierten Schlüssel von A
- S verwendet eine Version von PGP, die ADKs auch als Non-Hashed Subpackets akzeptiert und den Nutzer nicht informiert
- die verschlüsselte E-Mail an E enthält den Sitzungsschlüssel zum Entschlüsseln der Nachricht mind. zweimal:
  →einmal verschlüsselt mit dem öffentlichen Schlüssel von E
  →einmal mit dem von A
- wenn A Zugriff auf irgendein Element in die Kommunikationskette zwischen S und E hat, kann er die verschlüsselte Nachricht mitschneiden und mit seinem privaten Schlüssel entschlüsseln

Fazit: Angriff manipuliert den öffentlichen PGP-Schlüssel

Folge: einzelne mit dem manipulierten öffentlichen Schlüssel verschlüsselte Nachrichten konnten auch vom Angreifer gelesen werden

Wichtig:

- Folgerung, dass alle Verweise auf Public Keys als Hashed Subpacket eingefügt werden müssen
- Sicherheitstechnische Informationen in Nonhashed Subpackets müssen ignoriert oder besser als Fehler angezeigt werden.

Mathematische Grundlagen und Funktionsweise von RSA

Chinesischer Restsatz:

Seien $m_1, \ldots, m_2 \in \mathbb{Z}$ paarweise teilerfremd und seien a1,…, ar $\in \mathbb{Z}$ beliebig.
$\Rightarrow$ Das System $x \equiv a_1 \pmod{m_1}$
.
.
.
$x \equiv a_1 \pmod{m_r}$
besitzt eine Lösung b$\in \mathbb{Z}$ und jede weitere Lösung unterscheidet sich von dieser um ein Vielfaches von $m_1 \cdot \ldots \cdot m_r$,
d.h. Abb. $\mathbb{Z}_{/(m1\cdot\ldots\cdot mr)}\ \mathbb{Z} \rightarrow \mathbb{Z}/m_1\ \mathbb{Z}x \ldots x\mathbb{Z}/mr\ \mathbb{Z}$ bijektiv
$x \mapsto (x+m_1\mathbb{Z},\ldots,x+m_r\mathbb{Z})$

Satz von Euler

Sei a$\in \mathbb{Z}$ und m$\in \mathbb{N}_+$ mit ggT(a,m)=1 $\Rightarrow a^{\varphi(m)} \equiv 1 \pmod{m}$

RSA (verkürzt)

1. Wähle zwei „sehr große“ Primzahlen p und q. Berechne $n=p \cdot q$
2. Berechne $\varphi(n)= \varphi(p\cdot q)=(p-1)(q-1)$
   Wähle eine Zufallszahl e$\in \{1,\ldots, \varphi(n)\}$ mit ggT(e,n)=1.
3. Berechne d $\in \{1,\ldots, \varphi(n)\}$ mit $de \equiv 1 \pmod{\varphi(n)}$ [Erweit. Eukl. Alg.]
4. öffentlicher Schlüssel: Paar (n,e) privater Schlüssel: d
5. Verschlüsselung einer Nachricht m: $c=m^e \bmod n$
6. Entschlüsselung eines Geheimtextes c: $m'=c^d \bmod n$
   $m' \equiv c^d \equiv (m^e)^d\ m^{1+k\,\varphi(n)} \equiv m \cdot 1 \equiv 1 \pmod{n}$

Digitale Signatur mit RSA A will Nachricht *m* an B senden

1. A bildet Hashwert der Nachricht *m:* h=hash(m)
2. A berechnet die Signatur: $sig=h^{dA} \pmod{n}$
3. A sendet $(m, sig)^{eB}$ an B
4. B entschlüsselt $(m, sig)^{eB}$ mit d_B und erhält m , sig
5. B berechnet h'=hash(m) und vergleicht mit $sig^{eA}=h$

Angriff 2: Manipulation des privaten Schlüssels

Angriff: auf den privaten Schlüssel (verschlüsselt gespeichert) eines PGP-Nutzers, um diesen zu brechen

→hier nicht nur Aufdeckung einzelner Nachrichten, sondern Brechung des Schlüssels des Nutzers

Schlüsselpaar eines Benutzers: öffentlicher Schlüssel + privater Schlüssel in der Datei secring.skr gespeichert

Struktur eines sekring.skr-Datensatzes 3 Teile:

1. Public-Key-paket (Tag 6 oder 14)
2. Liste von Parametern, die benötigt werden, um mit Hilfe der Passphrase des Nutzers den privaten Schlüssel entschlüsseln zu können
3. verschlüsselter privater Schlüssel des Nutzers zusammen mit einer einfachen Prüfsumme

Beobachtung: die 3 Teile kryptographisch nicht miteinander verknüpft
→Angreifer könnte z.B. das Publig-Key Paket manipulieren, ohne dass dies bei der Benutzung des privaten Schlüssels auffallen würde

Ziel: Knacken des privaten Schlüssels für das RSA-/DAS-Verfahren
→Verwendung von bereits bekannten Angriffstechnicken

einzige Hürde: Schreibzugriff auf die Datei sekring.skr des Opfers zu erhalten

Schreibzugriff auf sekring.skr:

Schreibzugriff auf eine Datei: durch das Betriebssystem geregelt

- keine Zugriffskontrollen:
  z.B. bei MS-DOS, Windows 95/98
  →Hier ist es ausreichend, wenn der Angreifer sich Zugang zum Computer verschafft, auf dem sekring.skr abgelegt ist (z.B. über ein Schandprogramm wie Trojanisches Pferd,…).

- bei Zugriffskontrollen:
  Username/ Passwort-Login zur Erkennung des autorisieren Nutzers
  z.B. bei Unix, Windows NT

  - aber: schwächerer Mechanismus als der Passphrasen-Mechanismus von PGP
  - viele Angriffstools existieren, um Passwörter zu berechnen
  - Administrator/ Supervisor haben Zugriffsrechte auf alle Dateien

→viele Angriffe in Netzwerken zielen darauf ab, sich solche Rootrechte zu verschaffen

Es ist durchaus möglich, sich Schreibzugriff auf die Datei sekring.skr zu verschaffen. Der Aufwand hierzu ist viel geringer als der beim Angriff auf den kryptografischen Mechanismus von PGP.

Struktur des Datensatzes eines RSA-Schlüsselpaares

1 Byte	Version number	Public Key
4 Byte	Creation number	
1 Byte	Algorithm (RSA)	
?	Modulus n	
?	Exponent e	
1 Byte	String-to-key-usage (0xFF)	Param.
(1)	Symmetrical algorithm	
(1+1+8+1)	0x03 (iterated and salted string-to-key identifier); identifier of the hash algorithm (for SHA-1 it is 0x02); salt (random data, which are hashed together with the user’s passphrase and diversifies thus derived symmetrical key); the number of hashed octets of the data (the so-called “count”).	
(8-16)	Initialisation vector IV	
2+128	Prefix + exponent d	Private Key
2+64	Prefix + prime p	
2+64	Prefix + prime q	
2+64	Prefix + plnv	
2	checksum, arithmetic sum of previous octets (prefixes and numbers d, p, q, plnv) as plaintext, modulo 65536 (in version 4 encrypted, in version 3 not encrypted)	

RSA: Fault Analysis

Angriff von Boneh

- sorgte vor einiger Zeit in der Chipindustrie für Aufregung

Ziel: RSA-Algorithmus dazu bringen, einen Fehler bei der Berechnung einer Signatur zu machen
→Aus der fehlerhaften Signatur kann dann der private Schlüssel berechnet werden!

Angriff besonders einfach anwendbar:
wenn aus Performancegründen die Werte $m^d \bmod p$ und $m^d \bmod q$ für $n=pq$ getrennt berechnet werden und die beiden Teilergebnisse mit dem Chinesischen Restsatz zu einer Gesamtlösung zusammengesetzt werden:

- p und q sind teilerfremd →ggT(p,q)=1
- $\exists a, b \in \mathbb{Z}$ mit $1=ap+bq$ (erweiterter Eukl. Alg.)
- Sei $s_p \equiv m^d \pmod p$
  $s_p \equiv m^d \pmod q$

Nach Chinesischem Restsatz existiert eine Lösung

$$s \equiv s_p\, bq + s_p\, ap \pmod n,$$

wegen $bq \equiv 1 \pmod p$ und $ap \equiv 1 \pmod q$

Wenn hier nun eine der beiden Berechnungen modulo p oder q verfälscht wird, kann man die Faktorisierung von n wie folgt berechnen:

- wir provozieren einen Fehler bei der Berechnung von s_p
  - der fehlerhafte Wert sei s_p'
- mit $s \equiv s_p\, bq + s_p\, ap \pmod n$ und $s' \equiv s_p\, bq + s_p'\, ap \pmod n$ gilt dann $ggT(s-s',n) = ggT((s_p - s_p')bq, pq) = q$
- damit ist ein Primfaktor von n gefunden und der zweite ist p=n/q

---

Performancesteigerung: bei PGP tatsächlich verwendet,
siehe Grafik „Struktur des Datensatzes eines RSA-Schlüsselpaares“

eigentlich ausreichend: nur den privaten Exponenten d verschlüsselt abzuspeichern
aber: Im privaten Teil des Datensatzes werden auch die geheimen Werte p, q, $pInv=p^{-1} \bmod q$ (notwendig für die Anwendung des Chinesischen Restsatzes) gespeichert.

Brechung des privaten Schlüssels:

Kann man einen der Werte p, q, pInv verändern und wird mit diesem veränderten Wert die Signatur erstellt, so sind nach dem Prinzip von oben aus der fehlerhaften Signatur p oder q von berechenbar und damit kann der private RSA-Schlüssel gebrochen werden.

Wie kann man die Werte verändern trotz Verschlüsselung mit einer Blockchiffre?

Es gibt verschiedene Versionen der Datenstruktur:

Version 3: Hier sind **nicht** verschlüsselt:
- -Präfixe mit Längenangaben für die nachfolgenden Multiprecision Integer MIP
- -einfache Prüfsumme über alle Bytes (als positive Zahlen aufgefasst) modulo 65536

Beispiel: Veränderung von pInv:
- -durch Veränderung der Längenangabe für pInv von 512 auf 511Bit
  →PGP-Software müsste das 512-te Bit ignorieren
- -Prüfsumme müsste nur um den gleichen Betrag verändert werden wie die Längenangabe

Gründe für den Erfolg der Angriffe

- Die beiden Teile des Schlüsselpaares sind in der Datenstruktur syntaktisch nicht verknüpft. Prüfsumme wird nur über den privaten Schlüssel gebildet.
- Die Verwendete Prüfsumme ist kryptographisch völlig ungeeignet.

Vorschläge zur Abwehr der Angriffe

Klima und Rosa machten mehrere Vorschläge zur Abwehrung der Angriffe:

- mathematische Prüfungen, die jede PGP-Implementierung selbst vornehmen kann, um die semantische Verknüpfung der beiden Teile zu überprüfen
- Vorschläge, wie die Datenstruktur des Schlüsselpaares im OpenPGP-Standard verbessert werden kann

Verwendete Literatur

- „Kryptografie in Theorie und Praxis“
  A. Beutelsbacher, H.B. Neumann, T. Schwarzpaul
- „Sicherheit und Kryptographie im Internet“
  Jörg Schwenk
- „Sicherheitskonzepte für das Internet“
  Martin Raepple
- „IT-Sicherheit“

Internetadressen

- www.pgp.com
  PGP herunterladen
- www.itl.nist.gov/fipspubs/fip180-1.html
  Informationen über SHA-1
- http://eprint.iacr.org/2002/076.pdf
  Vlastimil Klima und Tomas Rosa: Attack on Private Signature Keys of the OpenPGP format, PGP TM programs and other applications compatible with OpenPGP