Kryptografische Verfahren für Nicht-Mathematiker

Karsten Ness <k.ness@heinlein-support.de>

Inhaltsübersicht

- 1. Krypto-Primitive
  - Symmetrische Verschlüsselung
  - Public Key Verschlüsselung
  - DH-Schlüsseltausch
  - Elliptische Kurven
  - Post-Quanten-Crypto
  - Hash-Verfahren
- 2. Krypto-Protokolle
  - SSL/TLS
  - OpenSSH
- 3. Konfiguration
  - Webserver & Mailserver
  - SSH-Server & SSH-Client
- 4. Diskussion

Karsten Ness <k.ness@heinlein-support.de>

1: Symmetrische Verschlüsselung

- Wird seit mehr als 2.000 Jahren der verwendet.
- Ver- und Entschlüsselung verwenden den gleichen Key.
- Mathematische Operation ist umkehrbar (symmetrisch).

1: Symmetrische Verschlüsselung

- Wird seit mehr als 2.000 Jahren der verwendet.
- Ver- und Entschlüsselung verwenden den gleichen Key.
- Mathematische Operation ist umkehrbar (symmetrisch).

- Beispiel: Caesar-Verschlüsselung mit dem Key **„PARTY“**

A	B	C	D	E	F	G	H	I	...
P	**A**	**R**	**T**	**Y**	B	C	D	E	...

Plaintext: „Morgen!“ <---> Ciphertext: „Jlodya!“

1.1 Symmetrische Krypto-Verfahren

- 3DES, Camelia, Cast5 (veraltet, werden ausrangiert)
- RC4 (von der NSA wahrscheinlich vollständig gebrochen)

Karsten Ness <k.ness@heinlein-support.de>

1.1 Symmetrische Krypto-Verfahren

- 3DES, Camelia, Cast5 (veraltet, werden ausrangiert)
- RC4 (von der NSA wahrscheinlich vollständig gebrochen)
- AES (Sieger der NIST Competition 2001)
  - AES128 für Stromchiffre mit häufig wechselnden Schlüsseln verwendbar
  - AES256 Mindeststandard für Verschlüsselung gespeicherter Daten

Karsten Ness <k.ness@heinlein-support.de>

1.1 Symmetrische Krypto-Verfahren

- 3DES, Camelia, Cast5 (veraltet, werden ausrangiert)
- RC4 (von der NSA wahrscheinlich vollständig gebrochen)
- AES (Sieger der NIST Competition 2001)
  - AES128 für Stromchiffre mit häufig wechselnden Schlüsseln verwendbar
  - AES256 Mindeststandard für Verschlüsselung gespeicherter Daten
- Blowfish (verbessert als Twofish) und Serpent
  - Ebenfalls erfolgreich bei der NIST Competition 2001 (2. und 3.Platz)
  - Häufig mit AES kombiniert für hohe Anforderungen (XOR-Verknüpfung)
- Chacha20-poly1305 (von D.J. Bernstein & Tanja Lange)
  - Selbst-authentifizierender Cipher

1.2 Betriebsmodi für symmetrische Verschl.

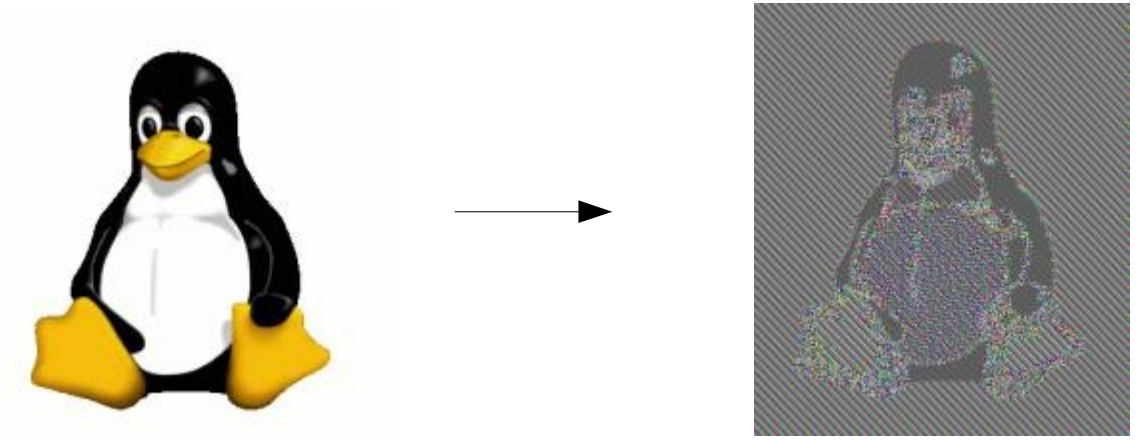
- Gleicher Input mit gleichem Key erzeugt identischen Output.
- Einfache symmetrische Verschlüsselung ist mit *Known-Plain-Text* Angriffen angreifbar.

1.2 Betriebsmodi für symmetrische Verschl.

- Gleicher Input mit gleichem Key erzeugt identischen Output.
- Einfache symmetrische Verschlüsselung ist mit *Known-Plain-Text* Angriffen angreifbar.
- Ein Beispiel:

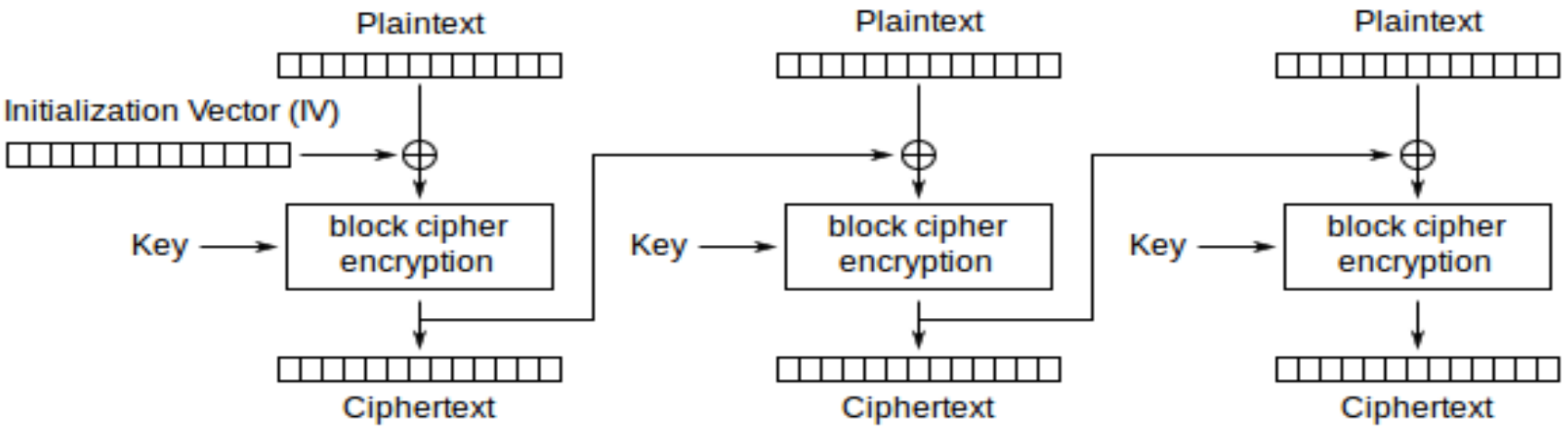
1.2 Betriebsmodi für symmetrische Verschl.

- Gleicher Input mit gleichem Key erzeugt identischen Output.
- Einfache symmetrische Verschlüsselung ist mit *Known-Plain-Text* Angriffen angreifbar.
- Ein Beispiel:



1.2 Betriebsmodi für symmetrische Verschl.

→ Gegenmaßnahme: ein Zufallsvektor wird mit verrechnet.



Cipher Block Chaining (CBC) mode encryption

Karsten Ness <k.ness@heinlein-support.de>

1.2 Betriebsmodi für symmetrische Verschl.

- Cipher Block Chaining (CBC)
  - Performant, viel genutzt, aber nicht mehr sehr sicher.
- Counter Mode (CTR)
  - Es wird zusätzlich ein Counter verwendet, der mit jedem Block hochzählt.
  - Damit ist die Datenmenge begrenzt, die verschlüsselt werden kann.
  - Für Stromchiffre mit geringer Bandbreite wie z.B. SSH verwendbar.
- Galois/Counter Mode (GCM)
  - Derzeitige Empfehlung für Stromchiffre wie TLS oder SSH.
- XEX-TCB-CTS (XTS, XTS64)
  - Derzeitige Empfehlung für Blockchiffre (Verschlüsselung von Blockdevices)

1.2 Betriebsmodi für symmetrische Verschl.

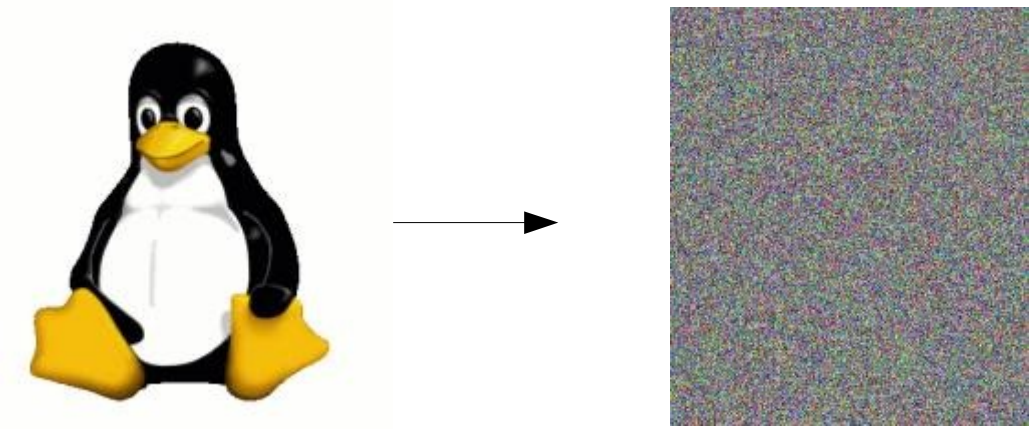
Bei symmetrischer Verschlüsselung ist immer der Betriebsmodus mit anzugeben, also:

AES128-CBC, AES256-GCM, Twofish-XTS...

1.2 Betriebsmodi für symmetrische Verschl.

Bei symmetrischer Verschlüsselung ist immer der Betriebsmodus mit anzugeben, also:

AES128-CBC, AES256-GCM, Twofish-XTS...



1.2 Public/Private Key Kryptografie

- Problem: Verteilung / Schutz der Schlüssel
- Ausweg: Asymmetrische Kryptografie
  - Public Key kann öffentlich verteilt werden.
  - Public Key wird zum Verschlüsseln verwendet.
  - Mit dem Private Key können die Daten dechiffriert werden.
- Mathematischen Anforderungen:
  - Verschlüsslung nicht bzw. schwer umkehrbar (Einwegfunktion)

Karsten Ness <k.ness@heinlein-support.de>

1.2 Public/Private Key Kryptografie

- Problem: Authentifizierung einer Nachricht
- Ausweg: Asymmetrische Kryptografie
  - Public Key kann öffentlich verteilt werden.
  - Private Key wird für das Erstellen einer Signatur verwendet.
  - Mit dem Public Key kann die Signatur verifiziert werden.

Karsten Ness <k.ness@heinlein-support.de>

1.2 Public/Private Key Kryptografie

- Problem: Authentifizierung einer Nachricht
- Ausweg: Asymmetrische Kryptografie
  - Public Key kann öffentlich verteilt werden.
  - Private Key wird für das Erstellen einer Signatur verwendet.
  - Mit dem Public Key kann die Signatur verifiziert werden.

- Eine digitale Signatur mit einem Private Key beweist nur, dass der Absender die Hoheit über einen bestimmten Schlüssel hat. Sie beweist nicht die Identität des Absenders.
  - Um die Identität eines Absenders oder eines Servers zu verifizieren, nutzt man Zertifikate. (Ein Public Key ist kein Zertifikat!)

1.2 Public/Private Key Kryptografie

- RSA-Verfahren:
  - Verschlüsseln: $\mathbf{c = m^e \pmod N}$ mit Public Key = (e,N)
  - Entschlüsseln: $\mathbf{m = c^d \pmod N}$ mit Private Key = (d)

1.2 Public/Private Key Kryptografie

- RSA-Verfahren:
  - Verschlüsseln: $\mathbf{c = m^e \pmod N}$ mit Public Key = (e,N)
  - Entschlüsseln: $\mathbf{m = c^d \pmod N}$ mit Private Key = (d)
- Was hat das mit Primzahlen zu tun?

Karsten Ness <k.ness@heinlein-support.de>

1.2 Public/Private Key Kryptografie

- RSA-Verfahren:
  - Verschlüsseln: **$c = m^e \pmod N$** mit Public Key = (e,N)
  - Entschlüsseln: **$m = c^d \pmod N$** mit Private Key = (d)
- Rahmenbedingungen:
  - N ist das Produkt zweier großer Primzahlen **p** und **q**.
    - N hat 500-600 Stellen, ist schwer in seine Primzahlen zerlegbar.
  - (e) zufällig gewählt, muss teilerfremd zu (p-1)(q-1) sein.
  - (d) wird durch eine Funktion **f(e,p,q)** berechnet.
    - Angreifer kennt nur N, aber nicht p, q und kann d nicht berechnen.
    - Eleganter wäre e=f(d...), aber mathematisch leider nicht möglich.

1.2 Public/Private Key Kryptografie

Hybride Verschlüsselung

- Public/Private Key Krypto ist 1.000 mal langsamer als symmetrische Verschl.
- Daten werden mit einem zufällig generierten Key symmetrisch verschlüsselt.
- Der Key wird asymmetrisch verschlüsselt und mitgesendet.

1.2 Public/Private Key Kryptografie

Hybride Verschlüsselung

- Public/Private Key Krypto ist 1.000 mal langsamer als symmetrische Verschl.
- Daten werden mit einem zufällig generierten Key symmetrisch verschlüsselt.
- Der Key wird asymmetrisch verschlüsselt und mitgesendet.

Hybride Signatur

- Es wird ein Hashwert über die Daten berechnet.
- Der Hashwert wird mit dem privaten Schlüssel signiert.

Karsten Ness <k.ness@heinlein-support.de>

1.2 Zertifikate

- In God you may trust.
- If you want to use crypto,
  you have to be sure about the keys.

1.2 Zertifikate

Ein Zertifikat besteht ganz allgemein aus:

- Common Name (Servername, E-Mail Adresse, Jabber-ID...)
- Kommentarfelder (Name, Firma, Land....)
- Public Key (für RSA-Schlüssel: e,N)
- Hashwert über alle Datenfelder
- Kryptografische Signaturen über den Hashwert zur Beglaubigung
  - Kryptografische Signatur mit einem privaten Schlüssel.
  - Informationen zum Zertifikat des Signierenden für die Verifikation.

- Ein Zertifikat verbindet einen kryptografischen Schlüssel mit einer Identität („Common Name“) und bietet Beglaubigungen dafür an.

Karsten Ness <k.ness@heinlein-support.de>

1.2 Hardware Security Module (HSM)

- Ein Hardware Security Module (HSM) speichert den privaten Key und führt alle Krypto-Operationen mit dem privaten Key aus.
  - Einsatz in unsicheren Umgebungen möglich (Smartphone, fremde Rechner)
- Ein Hardware Security Module bindet den Private Key an eine Identität (Person, Rolle in einer Firma...)
- Neben einem Passwort (PIN) authentifiziert sich der Inhaber des HSM durch den physischen Besitz des Token.
- Beispiele:
  - PKCS11-Token
  - OpenPGP Smartcards

Karsten Ness <k.ness@heinlein-support.de>

1.3 Diffie-Hellman Schlüsseltausch

- Problem 1: Sicher verschlüsselte Kommunikation über einen unsicheren Kanal etablieren.
- Problem 2: (Perfect) Forward Secrecy für RSA & Co.

Karsten Ness <k.ness@heinlein-support.de>

1.3 Diffie-Hellman Schlüsseltausch

- Problem 1: Sicher verschlüsselte Kommunikation über einen unsicheren Kanal etablieren.
- Problem 2: (Perfect) Forward Secrecy für RSA & Co.
- Mathematischer Ansatz:

$$\left(a^x\right)^y \left(mod\ N\right) = \left(a^y\right)^x \left(mod\ N\right) = S$$

Karsten Ness <k.ness@heinlein-support.de>

1.3 Diffie-Hellman Schlüsseltausch

(1) Alice und Bob einigen sich auf die DH-Parameter **a** und **N**.

→ In der Client-Server Kommunikation fragt der Client den Server nach den Parametern. Server sollten die DH-Parameter regelmäßig wechseln!!!

Karsten Ness <k.ness@heinlein-support.de>

1.3 Diffie-Hellman Schlüsseltausch

(1) Alice und Bob einigen sich auf die DH-Parameter a und N.

→ In der Client-Server Kommunikation fragt der Client den Server nach den Parametern. Server sollten die DH-Parameter regelmäßig wechseln!!!

(2) Alice wählt einen geheimes x, berechnet p und schickt das Ergebnis an Bob: $\mathbf{a^x \bmod N = p}$

Bob wählt ein geheimes y, berechnet q und schickt das Ergebnis an Alice: $\mathbf{a^y \bmod N = q}$

Karsten Ness <k.ness@heinlein-support.de>

1.3 Diffie-Hellman Schlüsseltausch

(1) Alice und Bob einigen sich auf die DH-Parameter **a** und **N**.

→ In der Client-Server Kommunikation fragt der Client den Server nach den Parametern. Server sollten die DH-Parameter regelmäßig wechseln!!!

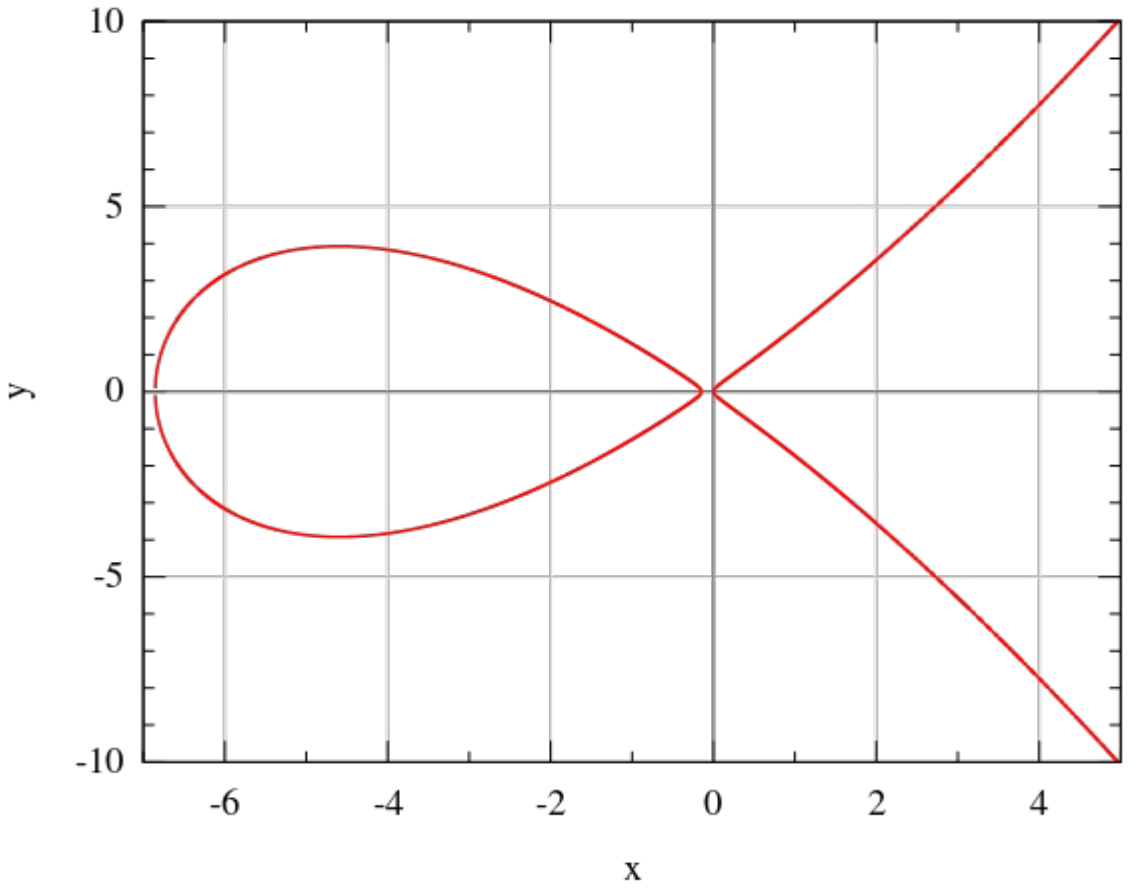
(2) Alice wählt einen geheimes x, berechnet p und schickt das Ergebnis an Bob: $\mathbf{a^x \bmod N = p}$

Bob wählt ein geheimes y, berechnet q und schickt das Ergebnis an Alice: $\mathbf{a^y \bmod N = q}$

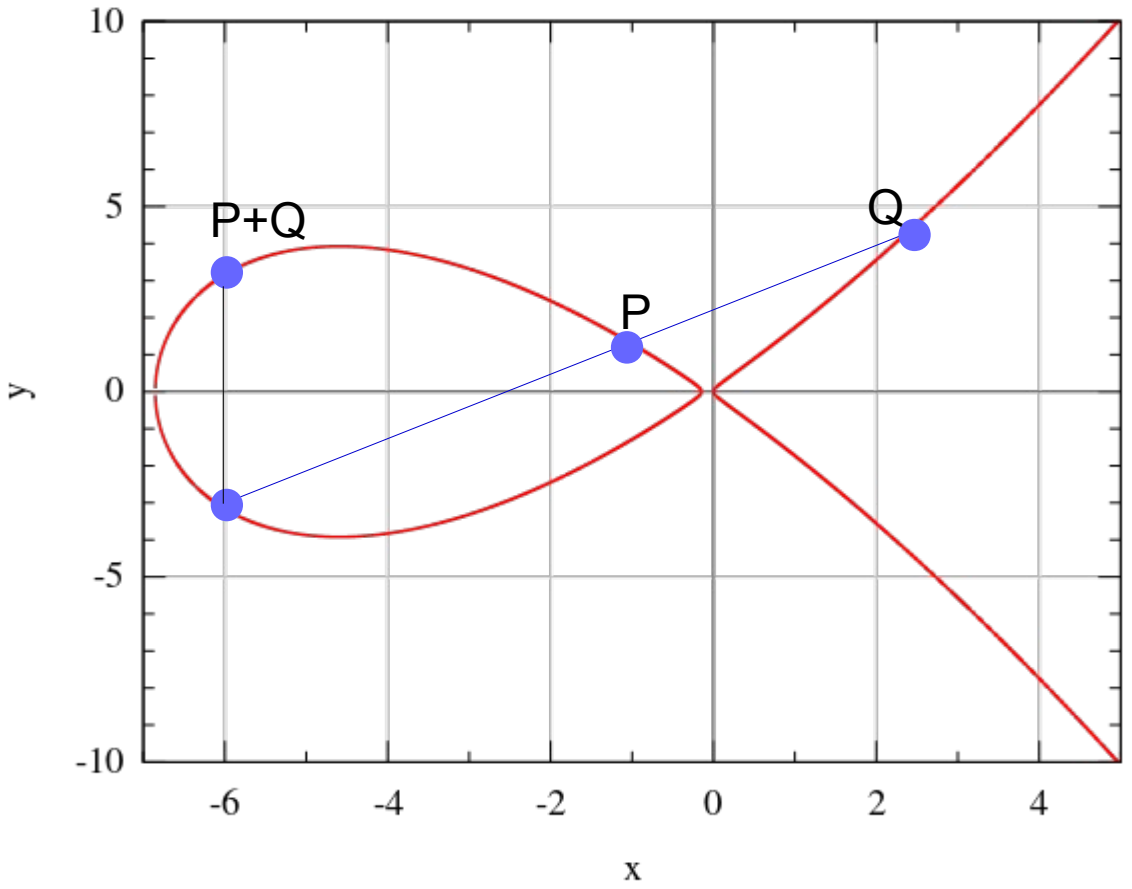
→ Beide können den geheimen Schlüssel S berechnen:

$$\mathbf{p^y \bmod N = q^x \bmod N = S}$$

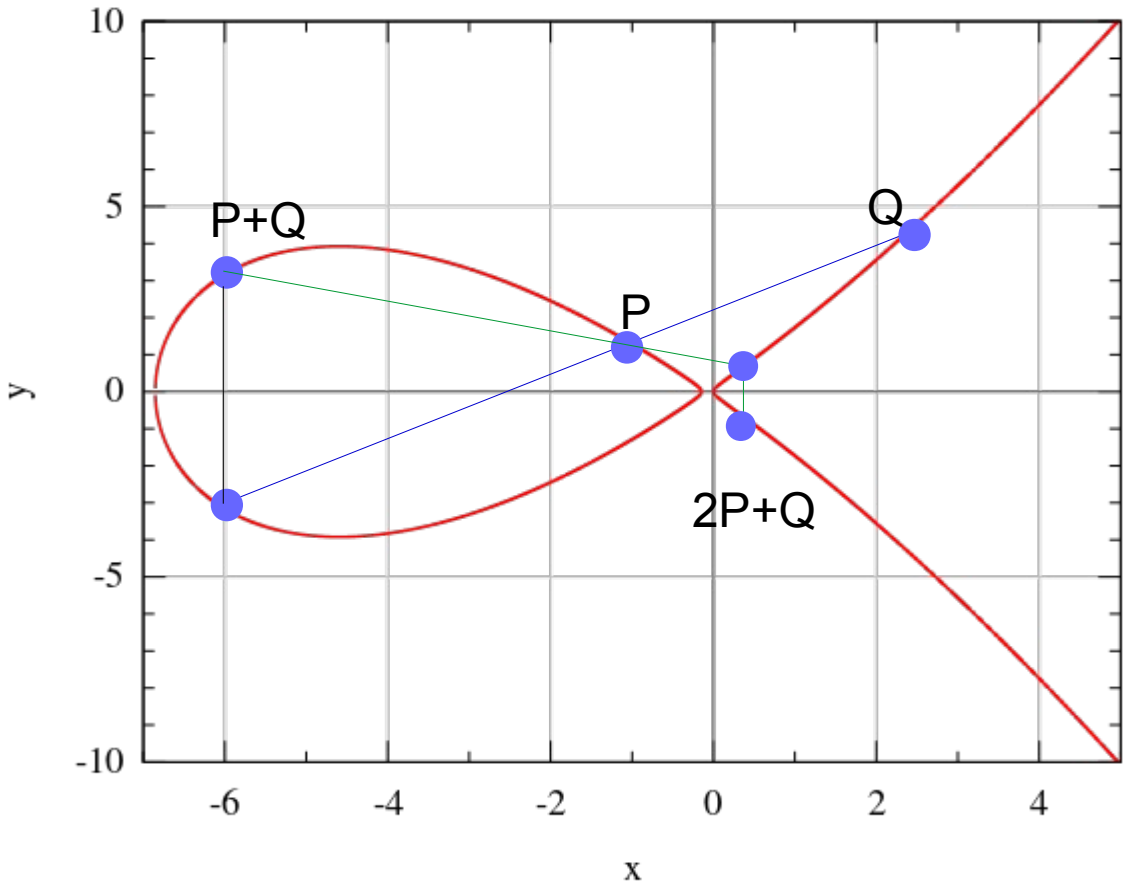
1.4 Kryptografie mit elliptischen Kurven



1.4 Kryptografie mit elliptischen Kurven



1.4 Kryptografie mit elliptischen Kurven



1.4 Kryptografie mit elliptischen Kurven

- Es wird eine Multiplikation **R = n*P+Q** definiert.
- Umkehrung **$P = n^{-1} * (R-Q)$** ist schwer lösbar
  - Es muss (wie bei RSA) ein Diskreter Logarithmus gelöst werden, wofür nur ineffiziente, langsame Verfahren bekannt sind.
- Kann für asymmetrische Kryptografie eingesetzt werden.

- Vorteile: kleinere Schlüssel als RSA, hohe Performance

1.4 Kryptografie mit elliptischen Kurven

- Kurven-Parameter:
  - NIST-Kurven wurden von der NSA geliefert (Suite B)
  - ECC-Brainpool Kurven werden z.B. in DE für hoheitliche Aufgaben genutzt.
  - Curve25591 (Bernstein/Lange) wird in Tor, OpenSSH, DNScrypt... eingesetzt.

Karsten Ness <k.ness@heinlein-support.de>

1.5 Post-Quanten-Crypto

- Es gibt die theor. Vermutung, das Quanten-Computer die Primzahlzerlegung und das Diskrete Logarithmus Problem einfach lösen können. (Peter W. Shor, 1994)

Karsten Ness <k.ness@heinlein-support.de>

1.5 Post-Quanten-Crypto

- Es gibt die theor. Vermutung, das Quanten-Computer die Primzahlzerlegung und das Diskrete Logarithmus Problem einfach lösen können. (Peter W. Shor, 1994)
- Damit wäre es möglich, RSA, ECC usw. zu brechen.

Karsten Ness <k.ness@heinlein-support.de>

1.5 Post-Quanten-Crypto

- Es gibt die theor. Vermutung, das Quanten-Computer die Primzahlzerlegung und das Diskrete Logarithmus Problem einfach lösen können. (Peter W. Shor, 1994)
- Damit wäre es möglich, RSA, ECC usw. zu brechen.
- Robust gegen Quanten-Computer Angriffe sind z.B.
  - NTRU (Public/Private Key Kryptografie, stabil, patent-abhängig)
  - Hidden-Goppa-Code-Crypto (noch wenig erforscht)
  - ....

Karsten Ness <k.ness@heinlein-support.de>

1.6 Hash-Verfahren

- Ein Hash ist eine Prüfsumme für ein Datenpaket.
- Beispiel: Prüfsumme für eine Zahlenfolge:
  - Zahlfolge: {8,12,32,48,207}
  - Prüfsumme: (8+12+32+48+207) mod 16 = 3

Karsten Ness <k.ness@heinlein-support.de>

1.6 Hash-Verfahren

- MD4, MD5 (gebrochen)
- RIPEMD-160
- SHA1 (schwächelt, sollte nicht mehr genutzt werden)
- SHA2 (SHA256, SHA384, SHA512, verbesserte Verfahren)

Karsten Ness <k.ness@heinlein-support.de>

1.6 Hash-Verfahren

- MD4, MD5 (gebrochen)
- RIPEMD-160
- SHA1 (schwächelt, sollte nicht mehr genutzt werden)
- SHA2 (SHA256, SHA384, SHA512, verbesserte Verfahren)

- Keccak (Sieger der NIST SHA3-Competition)
  - Gut: weil leicht in Hardware implementierbar
  - Schlecht: weil leicht in Hardware implementierbar
- Skein (erfolgreich bei NIST SHA3-Competition)

1.6 Hash Message Authentication Code

- Bei der Berechnung des Hashwertes wird zusätzlich zu den Daten ein geheimer Schlüssel verwendet.
  - Vorteil: Das Hash-Verfahren muss nicht robust gegen Kollisionen sein
  - Nachteil: Der geheime Schlüssel muss bei der Berechnung und bei der Verifikation bekannt sein, darf aber nicht einem Angreifer bekannt sein.

1.6 Hash Message Authentication Code

- Bei der Berechnung des Hashwertes wird zusätzlich zu den Daten ein geheimer Schlüssel verwendet.
  - Vorteil: Das Hash-Verfahren muss nicht robust gegen Kollisionen sein
  - Nachteil: Der geheime Schlüssel muss bei der Berechnung und bei der Verifikation bekannt sein, darf aber nicht einem Angreifer bekannt sein.

- Verfahren:
  - HMAC-MD5
  - HMAC-SHA
  - HMAC-SHA256

Hash-Verfahren für Passwörter

- Bcrypt
- PBKDF2

Mehrfache Anwendung erschwert Brute-Force-Angriffe zusätzlich.

- Veracrypt wendet PBKDF2-RIPEMD160 insgesamt 327.661 mal an.

„Salzen“ von Hashes erschwert Angriffe mit Rainbow Tables.

Karsten Ness <k.ness@heinlein-support.de>

2.1 SSL/TLS Protokoll

- Empfehlungen der IETF für SSL/TLS:
  - Protokolle:
    - SSLv3 darf nicht mehr verwendet werden
    - TLS 1.0 / 1.1 sollten nicht mehr verwendet werden.
    - **TLS 1.2** wird empfohlen.
  - Empfohlene Cipher:
    - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

2.1 SSL/TLS Protokoll

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

1) Schlüsselaustausch mit ECDHE (Alternative: DHE mit 2048 Bit)
2) Authentifizierung des Server (und optional Client) mit RSA-Verfahren
3) Message Authentifizierung mit Hash SHA384 (Alternative: SHA256)
4) Verschlüsselung der Daten mit AES256-GCM (Alternative: AES128-GCM)

Karsten Ness <k.ness@heinlein-support.de>

2.1 SSL/TLS Protokoll

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

1) Schlüsselaustausch mit ECDHE (Alternative: DHE mit 2014 Bit)
→ Authentifizierung (Server und optional Client) mit RSA-Verfahren
→ Authentifizierung der Daten mit Hash SHA384 (Alternative: SHA256)
1) Verschlüsselung mit AES256-GCM (Alternative: AES128-GCM)

TLS_RSA_WITH_AES_128_CBC_SHA (nicht mehr empfohlen)

1) Authentifizierung und Session Key Übertragung mit RSA
2) Verschlüsselung der Daten mit AES128-CBC
3) Authentifizierung der Daten mit Hash SHA1

Karsten Ness <k.ness@heinlein-support.de>

1.2 TLS Downgrade Angriff

1. Thunderbird an Postfix:

   „Helo – ich hätte gern TLS 1.2 mit **tls-ecdhe-rsa-with-aes-128-gcm-sha256**."

2. Man-in-the-middle blockiert (1) und sendet an Postfix:

   „Helo – ich hätte gern TLS 1.0 mit **tls-rsa-with-rc4-128-sha**."

3. Postfix an Thunderbird:

   „Ehlo – dann einigen wir uns auf **tls-rsa-rc4-128-sha**."
   Stempel + Unterschrift

4. Thunderbird an Postfix:

   Uhhgg??? Stempel und Unterschrift sind korrekt, also gut – RC4.
   „Helo – na ja, wenn es nicht anders geht - starte Schlüsselübertragung mit RSA."

Karsten Ness <k.ness@heinlein-support.de>

1.2 TLS Downgrade Angrif

→ Schutzmaßnahmen:

1) Authentifizierung des Client.
2) Server akzeptiert keine schwachen Cipher.
3) Client akzeptiert keine schwachen Cipher.

Karsten Ness <k.ness@heinlein-support.de>

2.2 OpenSSH Protokoll 2

- Symmetrische Verschlüsselung:
  - Empfohlen: chacha20-poly1305 (ab Version 6.6), aes256-gcm, aes128-gcm
    Kompatibilitätsmodus: aes256-ctr, aes192-ctr, aes128-ctr, aes128-cbc
- Authentifizierung von Server & Client (Public/Private Keys)
  - RSA bis 4096 Bit, DSA (nicht mehr empfohlen), ECC (ed25591, ab Version 6.6)
  - Es werden keine beglaubigten Zertifikate genutzt sondern Fingerprint des Key.
  - Passwort Authentifizierung für Clients möglich aber deaktivierbar.
- Hashes für Message Authentifizierung
  - Hash Message Authentication Code (HMAC) mit MD5 ... SHA512
- Schlüsseltausch
  - DHE oder ECDHE mit Curve25519 (ab Version 6.6)

3 Server Konfiguration

- Möglichst aktuelle Software verwenden
  - Debian 6.0: kein TLS 1.2 durchgängig (völlig unbrauchbar aus Sicht der Kryptografie)
  - Debian 7.0: DH-Param. im Apache max. 1024 Bit. OpenSSH ohne ECC-Keys, GnuTLS ohne DANE
  - Debian 8.0: GnuPGP kennt keine ECC-Keys

  - Upgrades sind in begrenztem Maß über Backports oder Sourcen möglich.

Karsten Ness <k.ness@heinlein-support.de>

3 Server Konfiguration

- Möglichst aktuelle Software verwenden
  - Debian 6.0: kein TLS 1.2 durchgängig (völlig unbrauchbar aus Sicht der Kryptografie)
  - Debian 7.0: DH-Param. im Apache max. 1024 Bit. OpenSSH ohne ECC-Keys, GnuTLS ohne DANE
  - Debian 8.0: GnuPGP kennt keine ECC-Keys

  - Upgrades sind in begrenztem Maß über Backports oder Sourcen möglich.

- Server-Applikationen sicher konfigurieren
  - Crypto-Libs unterstützen die Standards, Konfiguration erfolgt in den Anwendungen.
  - Nur als sicher empfohlenen Cipher freigeben
  - Für Kompatibilität mit alten Clients einzelne Legacy-Cipher freigeben
  - Alles andere verbieten

3 Server Konfiguration

- Apache (in ssl.conf konfigurieren)

  SSLProtocol all -SSLv3

  SSLHonorCipherOrder on

  SSLCipherSuite 'ECDHE-RSA-AES128-GCM-SHA256 : DHE-RSA-AES128-GCM-SHA256 : ECDHE-RSA-AES256-CBC-SHA256 : AES256 : DES-CBC3-SHA : !aNULL:!eNULL:!LOW:!MD5:!EXP:!PSK:!DSS:!DSA!!RC4:!SEED:!MEDIUM'

- Postfix

  smtpd_tls_exclude_ciphers = aNULL, eNULL, EXPORT, DES, RC4, MD5, PSK, aECDH, EDH-DSS-DES-CBC3-SHA, EDH-RSA-DES-CDC3-SHA, KRB5-DE5, CBC3-SHA

  smtpd_tls_dh1024_param_file = /etc/postfix/dh_2048.pem

3 Server Konfiguration

- Dovecot:

  ssl_prefer_server_ciphers = yes

  ssl_dh_parameters_length = 2048

  ssl_cipher_list=ECDHE-RSA-AES128-GCM-SHA256 : DHE-RSA-AES128-GCM-SHA256: ..... DES-CBC3-SHA : !aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS...

- OpenSSH:

  Ciphers chacha20-poly1305@openssh.com, aes256-gcm@openssh.com

  MACs hmac-sha2-512-etm@openssh.com

  KexAlgorithms curve25519-sha256@libssh.org, diffie-hellman-group-exchange-sha256

Alle Beispiele sind gekürzt und unvollständige Listen!!!

Die Cipher sind dem aktuellen Stand der Forschung anzupassen.

Soweit, so gut.

Fragen und Diskussionen!

Karsten Ness <k.ness@heinlein-support.de>

heinlein

Heinlein Support hilft bei allen Fragen rund um Linux-Server

HEINLEIN AKADEMIE

Von Profis für Profis: Wir vermitteln die oberen 10% Wissen: geballtes Wissen und umfangreiche Praxiserfahrung.

HEINLEIN HOSTING

Individuelles Business-Hosting mit perfekter Maintenance durch unsere Profis. Sicherheit und Verfügbarkeit stehen an erster Stelle.

HEINLEIN CONSULTING

Das Backup für Ihre Linux-Administration: LPIC-2-Profis lösen im CompetenceCall Notfälle, auch in SLAs mit 24/7-Verfügbarkeit.

HEINLEIN ELEMENTS

Hard- und Software-Appliances und speziell für den Serverbetrieb konzipierte Software rund ums Thema eMail.