Kryptografische Verfahren für Nicht-Mathematiker

- **Heinlein Support**
  - IT-Consulting und 24/7 Linux-Support mit ~28 Mitarbeitern
  - Eigener Betrieb eines ISPs seit 1992
  - Täglich tiefe Einblicke in die Herzen der IT aller Unternehmensgrößen

- 24/7-Notfall-Hotline: 030 / 40 50 5 – 110
  - 28 Spezialisten mit LPIC-2 und LPIC-3
  - Für alles rund um Linux & Server & DMZ
  - Akutes: Downtimes, Performanceprobleme, Hackereinbrüche, Datenverlust
  - Strategisches: Revision, Planung, Beratung, Konfigurationshilfe

Inhaltsübersicht

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

1.1 Symmetrische Verschlüsselung

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

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

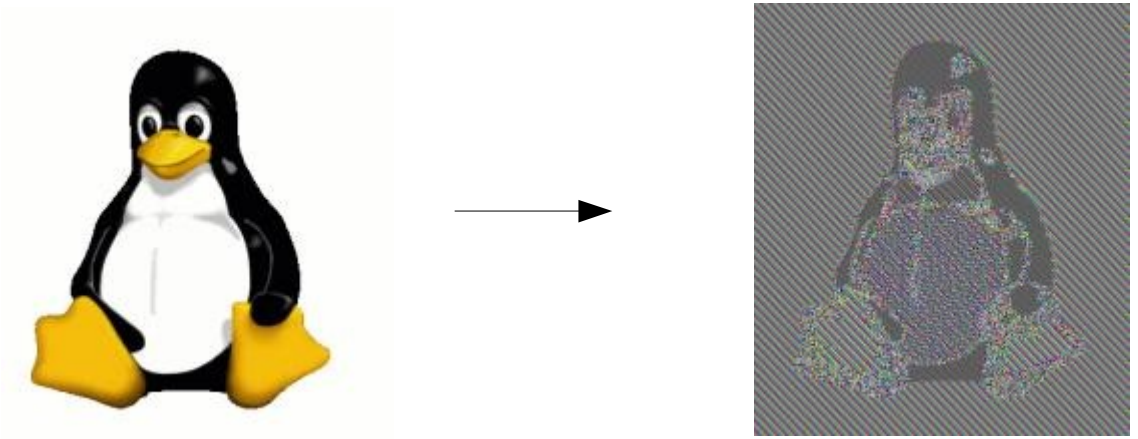
A	B	C	D	E	F	G	H	I	...
P	**A**	**R**	**T**	**Y**	B	C	D	E	...

1. Krypto-Primitive 1.1 Symmetrische Verschlüsselung

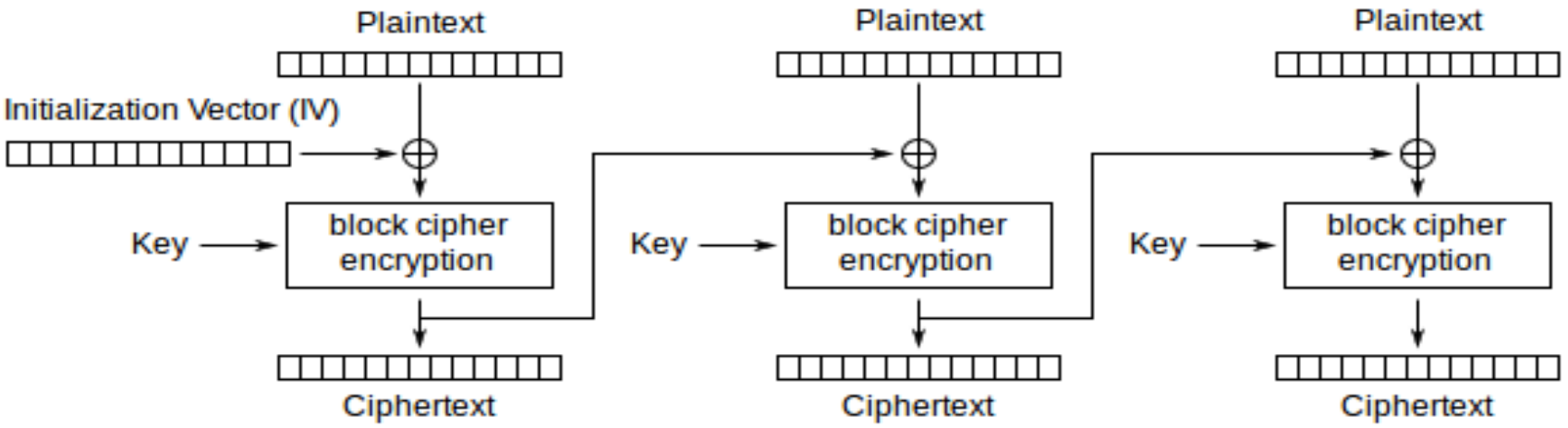
- 3DES, Camellia, Cast5 (veraltet, werden ausrangiert)
- RC4 (schnell aber schwach, von der NSA vollständig gebrochen?)
- AES (Sieger der NIST Competition 2001)
  - AES128 für Stromchiffre mit häufig wechselnden Session Keys verwendbar
  - AES256 Mindeststandard für Verschlüsselung gespeicherter Daten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

Betriebsmodi für symmetrische Verschlüsselung

- Gleicher Input mit gleichem Key erzeugt identischen Output.
- Angreifbar mit *Known-Plain-Text* Angriffen
- Ein Beispiel:



→ Gegenmaßnahme: ein Zufallsvektor wird mit verrechnet.



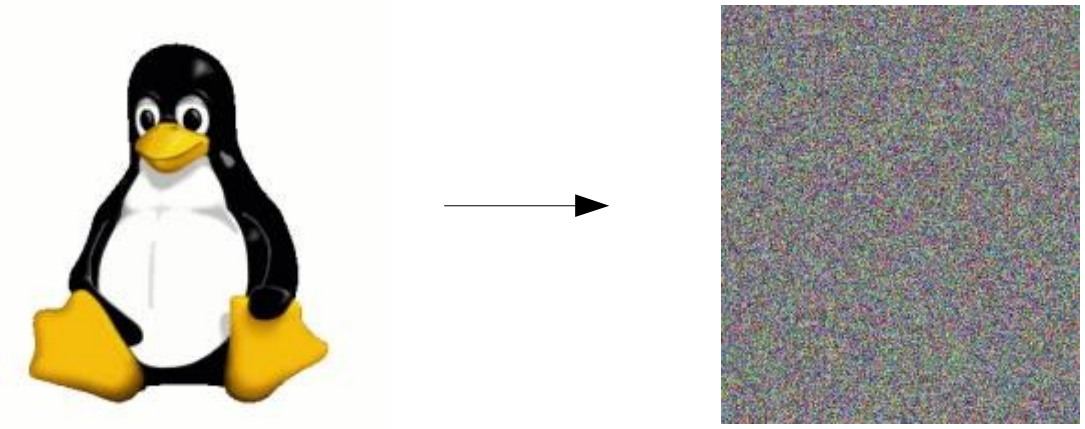
Cipher Block Chaining (CBC) mode encryption

1. Krypto-Primitive – 1.2 Symmetrische Verschlüsselung: Betriebsmodi

- 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)

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

→ AES128-CBC, AES256-GCM, Twofish-XTS…



1.2 Asymmetrische Verschlüsselung

- 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)

1.2 Asymmetrische Verschlüsselung

- 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.

1.2 Asymmetrische Verschlüsselung

- 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!)

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

- 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?

1. Krypto-Primitive 1.2 Asymmetrische Verschlüsselung

- 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)
- 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.

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 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.

Zertifikate

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

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.

- Beglaubigungen werden bei SSL/TLS, S/MIME und OpenPGP genutzt
- Andere Protokolle wie OTR, SSH, OMEMO, Acxolotl verzichten auf diesen Bullshit

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 (z.B. NitroKey oder Yubikey Neo für PGP und SSH)

1.3 Diffie-Hellman Schlüsseltausch

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

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:

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

1. Krypto-Primitive 1.3 Diffie-Hellmann 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!!!

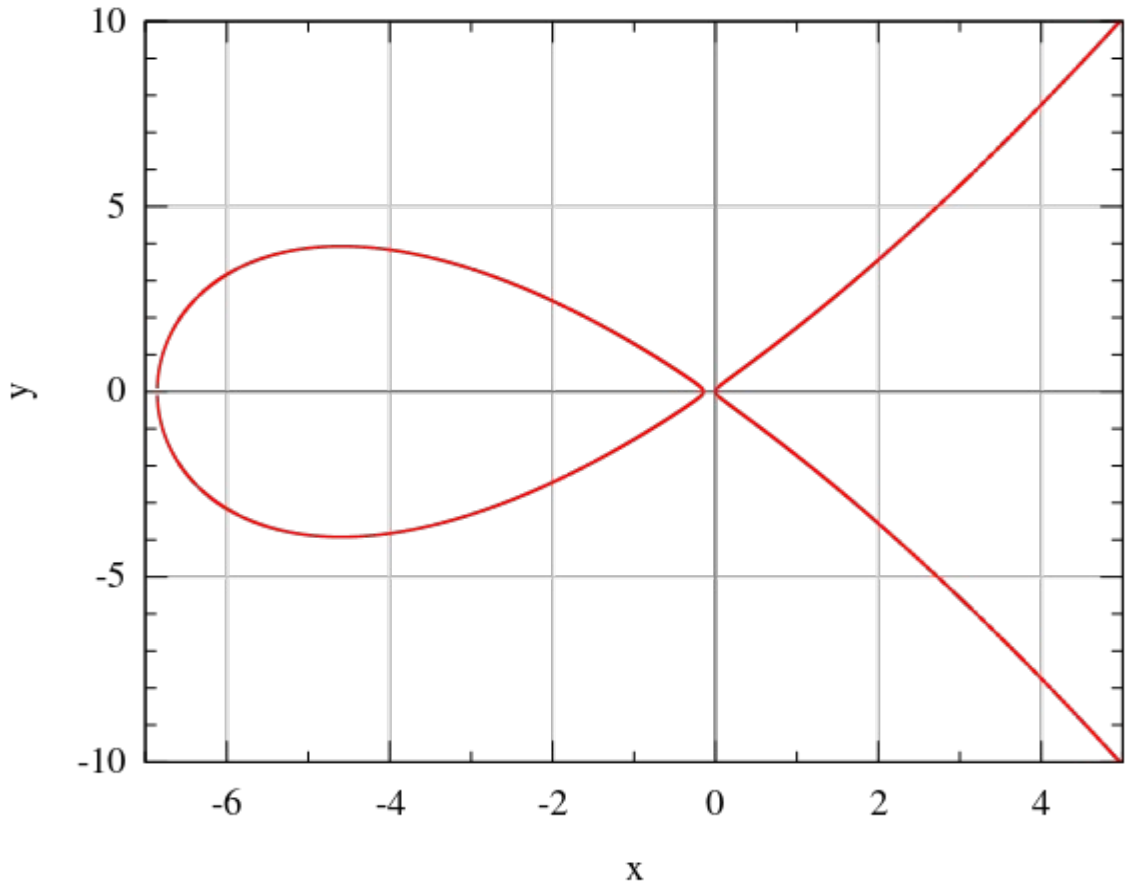
(2) Alice wählt einen geheimes x, berechnet p und schickt das Ergebnis an Bob: $\mathbf{a^x \bmod N = p}$

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

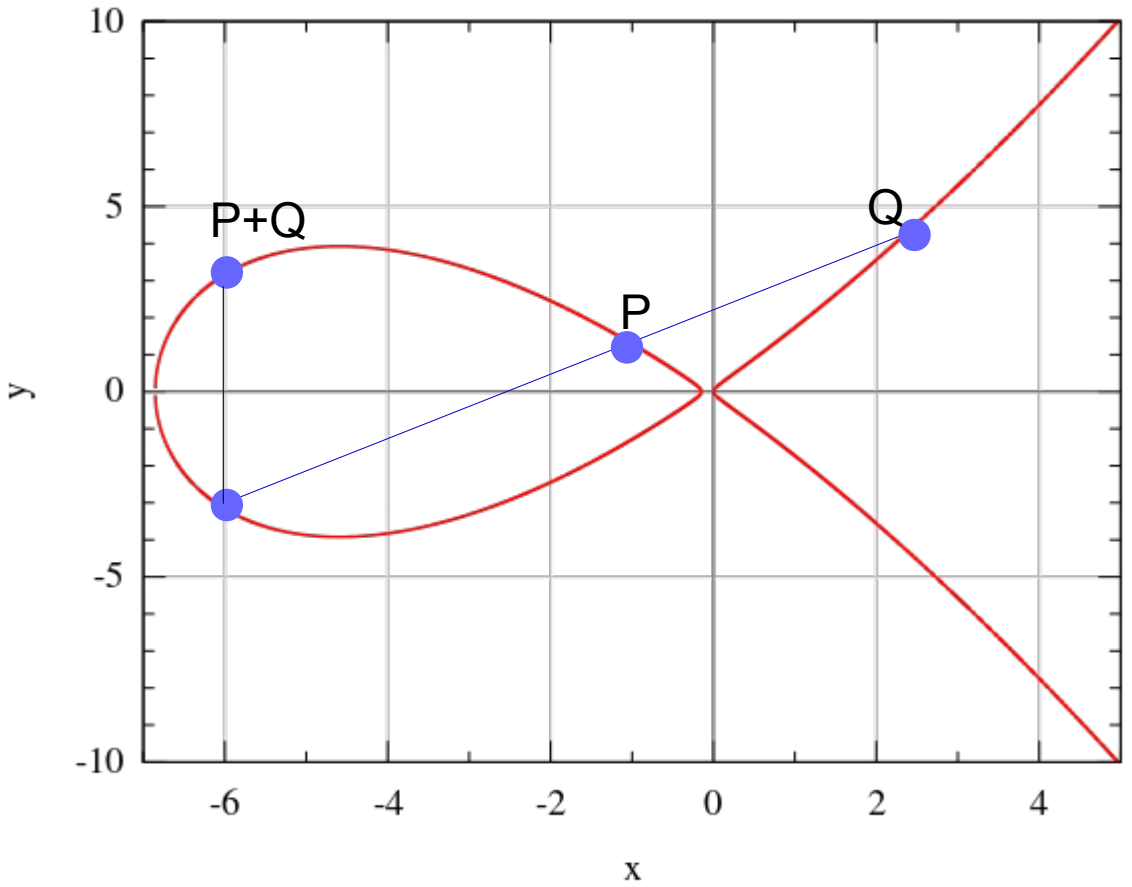
→ Beide können den geheimen Schlüssel S berechnen:

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

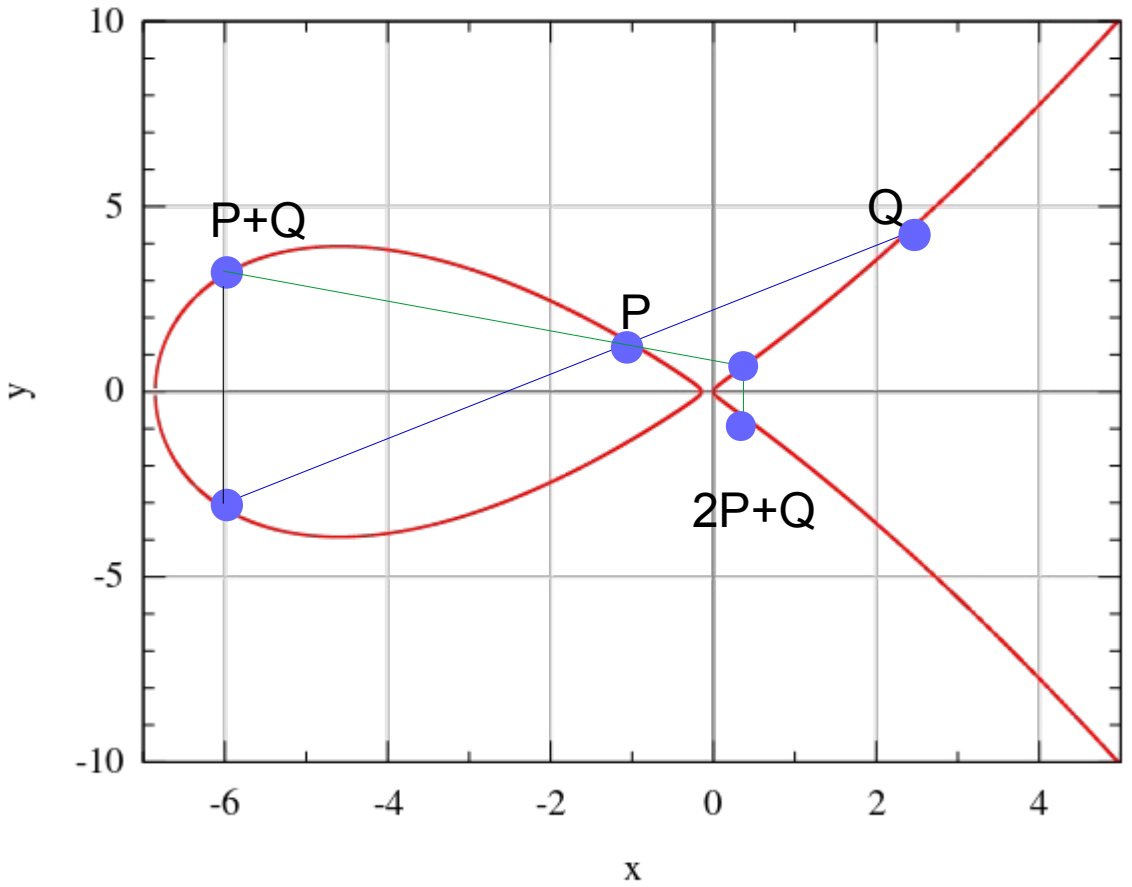
1.4 Kryptografie mit elliptischen Kurven



1.4 Kryptografie mit elliptischen Kurven



1.4 Kryptografie mit elliptischen Kurven



1. Krypto-Primitive 1.4 Elliptische 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 deshalb für asymmetrische Kryptografie eingesetzt werden

- Vorteile gegenüber RSA:
  - kleinere Schlüssel → hohe Performance, kein Angriff über Primzahl Zerlegung
- Nachteile gegenüber RSA:
  - kleinere Schlüssel → weniger robust gegen Q-Computer, wenig erforscht, komplexer und deshalb anfälliger gegen Seitenkanalangriffe auf Implementierung

- 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
- Analysen zur Sicherheit bietet http://safecurves.cr.yp.to

1.5 Post-Quanten-Crypto

- Es gibt die theoretische Vermutung, das Quanten-Computer die Primzahlzerlegung und das Diskrete Logarithmus Problem einfach lösen können. (Peter W. Shor, 1994)
- Damit wäre es möglich, RSA, ECC u.ä. zu brechen.
  - Die Anzahl der nötigen Q-Bits zum Brechen von RSA und ECC hängt nur von der Länge der Schlüssel ab → RSA ist robuster gegen Quantencomputerangriffe als ECC!
- Robust gegen Quanten-Computer Angriffe sind z.B.
  - NTRU (Public/Private Key Kryptografie, stabil, patent-abhängig)
  - Hidden-Goppa-Code-Crypto (noch wenig erforscht)
  - ....

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

1. Krypto-Primitive 1.6 Hashverfahren

- 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)

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.

2 Krypto-Protokolle

- Transportverschlüsselung (z.B. TLS, SSH, VPN...)
  - Verschlüsselung einer Verbindung zwischen zwei Rechnern
  - Daten werden verschlüsselt und authentifiziert, aber nicht verschleiert
  - Mit Traffic Correlation Attacks angreifbar, da Struktur der Daten erhalten bleibt

2 Krypto-Protokolle

- Transportverschlüsselung (z.B. TLS, SSH, VPN...)
  - Verschlüsselung einer Verbindung zwischen zwei Rechnern (Teilstück des Weges)
  - Daten werden verschlüsselt und authentifiziert, aber nicht verschleiert
  - Mit Traffic Correlation Attacks angreifbar, da Struktur der Daten erhalten bleibt
- Ende-zu-Ende Verschlüsselung (z.B. PGP, OTR, OMEMO, ZRTP...)
  - Verschlüsselung der Kommunikation zwischen zwei Endpunkten (für gesamten Weg)

2 Krypto-Protokolle

- Transportverschlüsselung (z.B. TLS, SSH, VPN...)
  - Verschlüsselung einer Verbindung zwischen zwei Rechnern (Teilstück des Weges)
  - Daten werden verschlüsselt und authentifiziert, aber nicht verschleiert
  - Mit Traffic Correlation Attacks angreifbar, da Struktur der Daten erhalten bleibt
- Ende-zu-Ende Verschlüsselung (z.B. PGP, OTR, OMEMO, ZRTP...)
  - Verschlüsselung der Kommunikation zwischen zwei Endpunkten (für gesamten Weg)
- Onion Encryption (z.B. Tor Onion Router, JonDonym, Remailer)
  - Zwiebelartige mehrfach Verschlüsselung, Verschleierung durch feste Paketegröße
  - Jeder Knoten auf dem Weg kann eine „Schale“ öffnen und findet die nötigen Daten
  - Verbessertes Onion Routing: zusätzlich Mixing und Dummy Traffic zur Verschleierung

2 Krypto-Protokolle

- Transportverschlüsselung (z.B. TLS, SSH, VPN...)
  - Verschlüsselung einer Verbindung zwischen zwei Rechnern (Teilstück des Weges)
  - Daten werden verschlüsselt und authentifiziert, aber nicht verschleiert
  - Mit Traffic Correlation Attacks angreifbar, da Struktur der Daten erhalten bleibt
- Ende-zu-Ende Verschlüsselung (OpenPGP, OTR, OMEMO, ZRTP...)
  - Verschlüsselung der Kommunikation zwischen zwei Endpunkten (für gesamten Weg)
- Onion Encryption (z.B. Tor Onion Router, JonDonym, Remailer)
  - Zwiebelartige mehrfach Verschlüsselung, Verschleierung durch feste Paketgröße
  - Jeder Knoten auf dem Weg kann eine „Schale“ öffnen und findet die nötigen Daten
  - Verbessertes Onion Routing: zusätzlich Mixing und Dummy Traffic zur Verschleierung
- Homomorphe Verschlüsselung (z.B. Wuala, SEAL...)
  - Soll das Rechnen mit verschlüsselten Daten ermöglichen, ohne dass Ausführende der Berechnung den Inhalt de Daten kennt

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)

2.1 SSL/TLS Protokoll

TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

- Schlüsselaustausch mit DHE (Alternative: ECDHE mit 2014 Bit)
- Authentifizierung (Server und optional Client) mit RSA-Verfahren
- Authentifizierung der Daten mit Hash SHA384 (Alternative: SHA256)
- 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

2. Krypto-Protokolle 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
- Empfehlungen des BSI für SSL/TLS
  - Es darf ausschließlich TLS 1.2 eingesetzt werden mit den dort definierten Ciphern
- NIST-Komptibilität

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."

How the NSA breaks so much crypto?

How the NSA breaks so much crypto?

- Die NSA hat die „most common DH primes“ geknackt
  - Wenn die „Sensoren“ der NSA einen Diffie-Hellmann Handshake erkennen, senden Sie den Handshake an einen Supercomputer
  - Dieser ermittelt innerhalb weniger Millisekunden den damit ausgehandelten Pre-Master Key und sendet das Ergebnis an den „Sensor“ (oder NULL, wenn nicht knackbar
  - Der „Sensor“ kann den weiteren Verlauf der TLS Session entschlüsseln und in Xkeyscore einspeisen

How the NSA breaks so much crypto?

- Die NSA hat die „most common DH primes“ geknackt
  - Wenn die „Sensoren“ der NSA einen Diffie-Hellmann Handshake erkennen, senden Sie den Handshake an einen Supercomputer
  - Dieser ermittelt innerhalb weniger Millisekunden den damit ausgehandelten Pre-Master Key und sendet das Ergebnis an den „Sensor“ (oder NULL, wenn nicht knackbar
  - Der „Sensor“ kann den weiteren Verlauf der TLS Session entschlüsseln und in Xkeyscore einspeisen
- Wenn Forward Secrecy mit DHE, dann richtig oder lieber garnicht!

Kryptographische Verfahren für Nicht-Mathematiker
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 mit Schlüsseln (üblich)
  - RSA bis 4096 Bit, DSA (nicht mehr empfohlen), ECC (ed25591, ab Version 6.6)
  - Es werden keine beglaubigten Zertifikate genutzt
  - Passwort Authentifizierung für Clients möglich aber unsicher.
  - High Security Level: Hardware Security Module nutzen (OpenPGP Cards, z.B. NitroKey)
- 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.

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

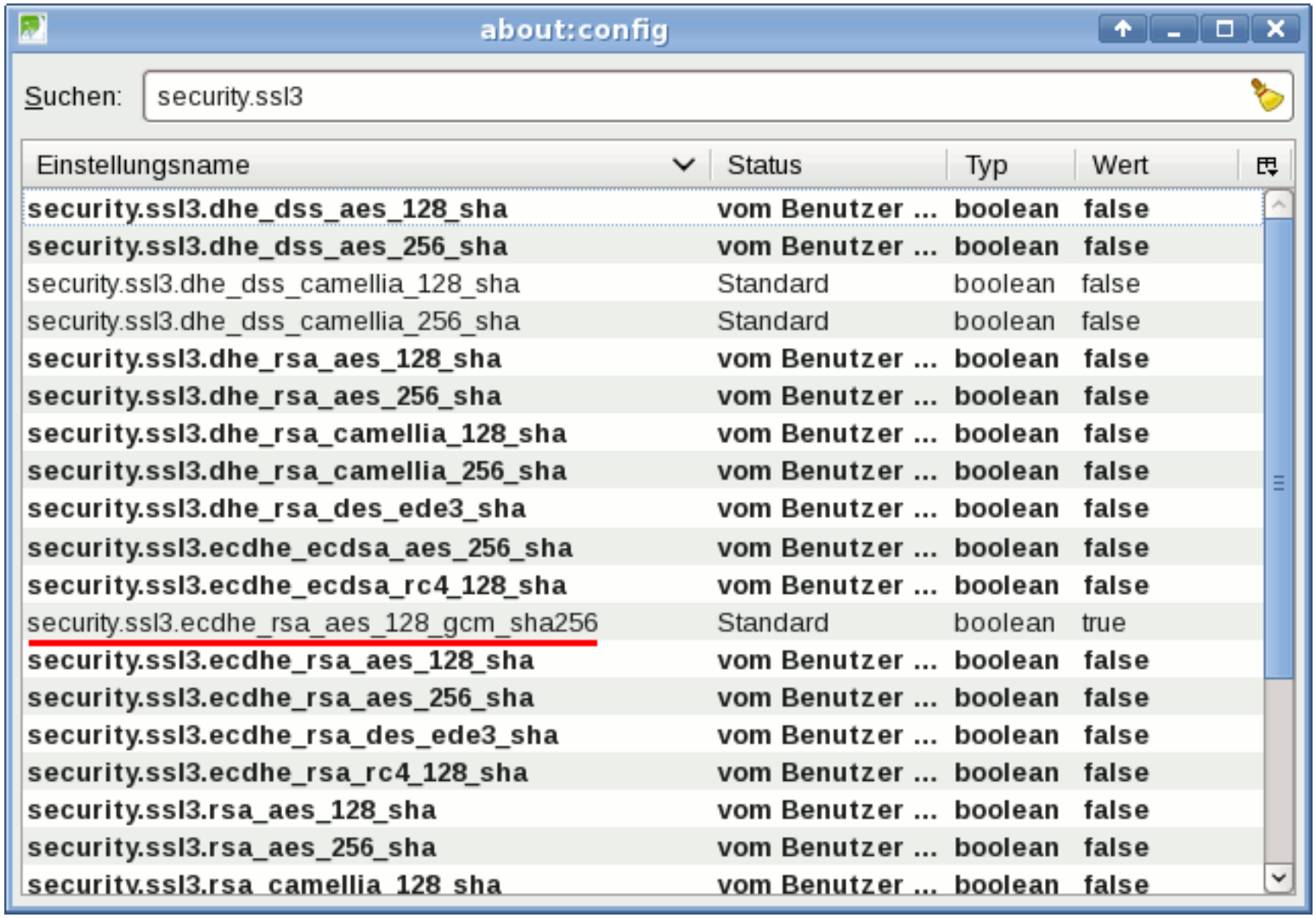
  - Upgrades sind in begrenztem Maß über Backports oder Sourcen möglich.

- 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 testen

- Für Webserver SSL Labs gut geeignet: https://www.ssllabs.com
- Für Mailserver gibt es z.B. CheckTLS: https://www.checktls.com
- Jabber Server testet das IM Repository: https://xmpp.net
- Allgm. Test auf NIST-Kompatibilität: https://www.htbridge.com/ssl

Client Konfiguration



Soweit, so gut.

Fragen und Diskussionen!

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.