Einführung Kryptographie & IT-Sicherheit

Andreas Uhl

Department of Artificial Intelligence and Human Interfaces (AIHI)
University of Salzburg

Somersemester 2022

Overview I

1. Formalia
2. Einleitung
   - Politik und Geschichte
   - Terminologie
   - Grundlagen: Einfache Methoden, Hash Funktionen, Protokolle
3. Klassische Kryptographische Algorithmen
   - DES
   - RSA
     - Mathematische Grundlagen: Zahlentheorie
     - RSA Algorithmus
   - Kecchak / SHA-3
4. Weitere Kryptographische Algorithmen
   - Stream Ciphers - Stromchiffren
   - AES
   - Weitere Public-Key Verschlüsselungs-Algorithmen
   - Hashfunktionen aus Block Ciphers und MACs

- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen
- Elliptic Curve Kryptographie Systeme
- Homomorphe Verschlüsselung
- Quantencomputer und Post-Quantum Cryptography

5 Netzwerksicherheit
- Sicherheit auf IP und DNS Ebene
- Sicherheit auf der Anwendungsebene

Lecturer

Email-Address: `uhl@cosy.sbg.ac.at`.

Basis-URL: `http://www.cosy.sbg.ac.at/~uhl`.

Office: FB Computerwissenschaften (Department of Computer Sciences), Room 1.15, Jakob-Haringer Str. 2, Salzburg-Itzling.

Telefon (Office): (0662) 8044-6303.

Telefon (Secretary): (0662) 8044-6328 or -6343.

When & Where

Course-URL:
`http://www.cosy.sbg.ac.at/~uhl/student.html.`

When: Mo 13:15 - 14:45

Interval: weekly

Where: Lecture Room T02, hybrid over webex
`https://uni-salzburg.webex.com/meet/andreas.u`

Organisatorisches & Prüfung

Willkommen zur VO “Grundlagen Kryptographie und IT-Sicherheit”. Durch die Vielzahl an wünschenswerten Inhalten sind VO und PS inhaltlich völlig getrennt. Um den Wegfall von Übungen zur VO etwas zu kompensieren gibt es in manchen Einheiten freiwillige HÜs, die vor der VO per e-Mail abgegeben werden müssen (So 18 Uhr) und auf Anfrage in der VO erklärt werden müssen (d.h. in diesem Fall Anwesenheitspflicht). Es gibt pro so einer abgegebenen HÜ einen Bonuspunkt für die Abschlussklausur (bei insgesamt 16 möglichen Punkten der schriftlichen Klausur).

2 Prüfungsvarianten:

1. “Normale” schriftliche Klausur mit 8 Fragen und 2 Punkten pro Frage.
2. On-line multiple-choice Test der bei positivem Bestehen 1 Punkt wert ist und die restlichen Punkte aus den HÜs zu einer positiven Note für die VO führen kann (dafür sind dann mindestens 7 solcher HÜs korrekt auszuarbeiten).

Literatur

Verfügbar UB digital

- A. Beutelspacher, J. Schwenk, K.-D. Wolfenstetter: Moderne Verfahren der Kryptographie (2015)
- C. Eckert: IT-Sicherheit (2018)
- R. Hellmann: IT-Sicherheit (2018)

- A. Beutelspacher, H. Neumann, T. Schwarzpaul: Kryptographie in Theorie und Proaxis (2010)
- J. Buchmann: Einführung in die Kryptographie (2004)
- B. Schneier: Applied Cyptography (1996)
- K. Schmeh: Kryptographie (2006)
- W. Stallings: Cryptography and Network Security (1999)
- D. Stinson: Cryptography - Theory and Practise (2002)

Top Journals

- Journal of Cryptology
- IEEE Transactions on Information Forensics and Security (TIFS)
- IEEE Transactions on Dependable and Secure Computing
- Computer & Security
- Security and Communication Networks
- Journal of Information Security and Applications
- IEEE Security and Privacy
- ACM Transactions on Security and Privacy

Internationale Konferenzen

- Crypto (IACR International Cryptology Conference), EuroCrypt, AsiaCrypt
- IEEE Symposium on Security and Privacy
- USENIX Security Symposium
- ACM Symposium on Computer and Communications Security
- IEEE Workshop on Information Forensics and Security (WIFS)
- IFIP International Information Security Conference, Information Security Conference (ISC), International Conference on Information and Communications Security (ISICS), International Conference on Information Security and Cryptology (ICISC)

- Privacy-protected Video Surveillance on Scalable Bitstreams (FFG, with Commend International, 200K EUR)
- Biometric Sensor Forensics (FWF, 280K EUR)
- Sample Data Compression and Encryption in Biometric Systems (FWF, 210K EUR)
- Metrics for Assessing Image and Video Encryption Schemes (FWF, 300K EUR)
- Pervasive and UseR Focused BiomeTrics BordEr ProjeCT (EU, 680K EUR)
- Finger-based Biometrics for Austrian ATMs (FFG/KIRAS, 220K EUR)
- Advanced Methods and Applications for Fingervein Recognition (FWF, 410K EUR)

Outline

1 Formalia

2 Einleitung
- Politik und Geschichte
- Terminologie
- Grundlagen: Einfache Methoden, Hash Funktionen, Protokolle

3 Klassische Kryptographische Algorithmen
  - Mathematische Grundlagen: Zahlentheorie
  - RSA Algorithmus

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit

Politik vs. Wissenschaft

Kryptographie wird seit Jahrtausenden benutzt, um mitlitärische und diplomatische Kommunikation zu schützen. Diese Tatsache führte zu der Annahme, Kryptographie sei ein Vorrecht der Regierung. Viele Regierungen üben heute eine Kontrolle auf die verwendeten kryptographischen Techniken - wenn nicht sogar auf die Forschung in diesem Bereich - aus. Ein Beispiel dafür sind die Export/Import Bestimmungen der USA für kryptographische Geräte. Diese entsprechen dem Gesetz für Waffenexport. Auch in der EU gibt es laufend Diskussionen über Beschränkung starker Kryptographie (Argument Terrorbekämpfung).

- 1929: Black Chamber - H.L. Stimson (U.S. secretary of state) "gentlemen do not read each others mail´´. Die kodierten diplomatischen Kanäle wurden routinemäßig gebrochen.
- 1940 wiedereröffnet, um die japanischen Verschlüsselungssysteme zu knacken.

Erst mit zunehmender Digitalisierung und digitaler Kommunikation entstand Bedarf nach Kryptographie im privaten und wirtschaftlichen Sektor.
Neben der Forschung im öffentlichen Bereich (Universitäten) gibt es Regierungslabors (z.B. NSA), die ihre Erkenntnisse selten publizieren und damit auch den Forschungsfortschritt in diesem Bereich behindern. In den USA müssen Regierungsstellen (oftmals geheime) NSA Verfahren verwenden und diese Verfahren werden auch privaten Stellen - unter NDA - zur Verfügung gestellt.

Durch das staatliche Interesse an der Thematik ergibt sich hier folgendes Problem: Kann bzw. soll ein(e) ForscherIn aus dem universitären Umfeld neue Erkenntnisse publizieren, die die eigene Regierung für gefährlich hält, z.B. nicht knackbare Verschlüsselungssysteme?

Geschichte I

Die Kryptographie kann in verschiedenen Epochen dargestellt werden.

inf $\rightarrow$ 1949: Diese Zeit kann als präwissenschaftliche Epoche bezeichnet werden und die Kryptographie ist eher eine Kunst denn eine Wissenschaft.
Julius Cäsar etwa verwendete folgendes System, mit z=3:

$$CAESAR \rightarrow FDHVDU$$

$$y = x \oplus z \qquad x.....A = 0, B = 1, ...Z = 25$$

z geheimer Schlüssel

y verschlüsselter Buchstabe

$\oplus$ Addition Modulo 26

Augustus verwendete das selbe System mit z=4, allg. als Shift-Cipher bezeichnet.

In dieser Epoche war die Kryptoanalyse der Kryptographie bei weitem überlegen.

Geschichte II

1926 G.S.Vernam entwickelte ein neues Verschlüsselungssystem, das sog. “one-time-pad” (OTP). Jedes Bit eines Textes x wird mit einem eigenem Schlüsselbit z verschlüsselt. Der codierte Text $y = x \oplus z$ (mit $\oplus$Addition Modulo 2, XOR) ist so nicht knackbar. Die Größe des Schlüssels entspricht allerdings der Größe des Textes.

Enigma & Co. Im zweiten Weltkrieg wurden erstmals MathematikerInnen in die kryptographische Forschung mit einbezogen, z.B. Turing in England, der eigene Entschlšselungshardware baut, die die deutsche Enigma knackt.

1949 C.E. Shannon veröffentlicht “Communication Theory of Secrecy Systems´´. Darin beweist er formal etwa die Unbrechbarkeit des Vernam Verfahrens. Dies hatte allerdings noch keinen großen Einfluß auf die wissenschaftliche Gemeinschaft.

Geschichte III

1976 W. Diffie und M.E. Hellman veröfftenlichen “New Directions in Cryptography´´. Dies enthält erstmals sogenannte Public Key Kryptographie. Dies sind Verfahren, die eine geheime Kommunikation ohne Transfer des Schlüssels zwischen Sender und Empfänger ermöglichen.

1984 A. Shamir bricht das Merkle-Hellman Trapdoor-Knapsack Public Key Kryptosystem.

2001 Die erste Ausführung des Shor Faktorisierungsalgorithmus auf einem Quantencomputer (IBM’s Almaden Research Center) könnte man als Beginn der Post-Quantum Cryptography Ära bezeichnen.

2015 J. Daemen gewinnt nach der AES Standardisierung auch den SHA-3 Wettbewerb.

2019 Google AI Quantum & NASA zeigen erstmalig Quantum Überlegenheit (schneller als Summit des Oak Ridge National Laboratory) auf dem Sycamore Quantencomputer.

Outline

1 Formalia

2 Einleitung
- Politik und Geschichte
- Terminologie
- Grundlagen: Einfache Methoden, Hash Funktionen, Protokolle

3 Klassische Kryptographische Algorithmen
  - Mathematische Grundlagen: Zahlentheorie
  - RSA Algorithmus

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit

Einführende Begriffe

Sender -Empfänger: Ein Sender möchte eine Nachricht so senden, daß ein potentieller Lauscher diese nicht lesen kann. Eine Nachricht ist ein Text, deren Inhalt zugänglich ist und heißt auch Plain Text oder Clear Text. Plain Text wird verschlüsselt (encryption), die verschlüsselte Nachricht heißt dann Ciphertext. Ciphertext muß vor dem Lesen entschlüsselt werden (decription) (encipher - decipher).

Kryptographie ist die Kunst bzw. Wissenschaft den Austausch von Nachrichten sicher zu machen.

Kryptoanalyse ist die Kunst bzw. Wissenschaft Kryptographische Methoden zu brechen.

Kryptologie ist ein mathematisches Teilgebiet, das sich mit Kryptographie und Kryptoanalyse beschäftigt. Die benötigten mathematischen Theorien hierfür stammen v.a. aus der Statistik und der Zahlentheorie.

Einführende Begriffe

Plaintext (M oder P) z.B. stream of bits, textfile, executable, ..., video, ... - am Computer sind das natürlich binäre Daten.

Ciphertext (C) ist das Ergebnis der Verschlüsselung des Plaintexts, das sind jedenfalls binäre Daten (mit state-of-the-art Ciphers), $\sharp bits(C) \geq \sharp bits(M)$.

Encription Function (E) E(M)=C

Decription Function (D) D(C)=M

$$\Rightarrow D(E(M)) = M!!$$

Mehr als Verschlüsselung !

- Geheimhaltung (confidentiality) wird erreicht durch Verschlüsselung unter der Annahme des freien Zugangs des Angreifers zum Kanal.
- Authentifizierung (Authentication) einer Person oder Nachricht - der Empfänger soll der Ursprung einer Nachricht sicher kennen und eine Person soll eindeutig indentifiziert werden.
- Datenunversehrtheit (Integrity) - ein Empfänger soll sicher sein können dass eine Nachricht nicht verändert worden ist.
- Nicht-Abstreitbarkeit (non-repudiation) - ein Sender/Empfänger soll nicht fälschlicherweise leugnen können, daß er eine Nachricht geschickt/erhalten hat.
- Anonymität (anonymity) - Identität des Senders oder Empfängers bzw. die Kommunikation soll verborgen werden.

Arten der Personen Authentifizierung

1. Passwort- oder PIN basierte Authentifizierung - durch etwas das man **weiss**. Probleme: kann vergessen oder an andere weitergegeben werden.
2. Token-basierte Authentifizierung - durch etwas das man **hat**, z.B. RFID-token, Schlüssel, Pass. Probleme: kann verloren oder an andere weitergegeben werden.
3. Biometrie-basierte Authentifizierung - durch etwas das man **ist**. Probleme: gegenwärtige Probleme haben Schwierigkeiten sobald ein biometrisches Merkmal kompromittiert wird, da es nicht ausgetauscht werden kann (Abhilfe: biometric Cryptosystems & cancelable biometrics).

Two-factor oder Multi-factor Authentifizierung kombiniert Methoden dieser verschiedenen Klassen, z.B. PIN und Smartphone.

Verwandte Begriffe

Privacy dt. Privatsphäre - Datenschutz, im engeren Sinn bedeutet es das Recht, Kontrolle über das Sammeln und Verarbeiten personenbezogener Daten zu haben, insbesondere dass persönliche Daten nur für den Zweck genutzt werden, für den sie gesammelt wurden.

Resilienz Fähigkeit von Systemen, bei Teil-Ausfällen oder Störungen nicht vollständig zu versagen, sondern wesentliche Systemdienstleistungen weiter aufrechtzuerhalten (Widerstandsfähigkeit).

Zuverlässigkeit ist geringe Ausfallshäufigkeit.

Verfügbarkeit - die Funktionen eines IT-Systems stehen ständig bzw. innerhalb einer vorgegebenen Zeit zur Verfügung und die Funktionalität des IT-Systems ist nicht vorübergehend oder dauerhaft beeinträchtigt.

Sicherheit Schutz von Systemen vor Schäden und Bedrohungen.

Verwandte Begriffe

IT-Sicherheit Schutz von IT-Systemen vor Schäden und Bedrohungen.

Cyber-Sicherheit Schutz von Cyber-physischen Systemen, IoT und Robotern vor Schäden und Bedrohungen.

Informationssicherheit IT-Sicherheit ist nur ein Teilaspekt der Informationssicherheit, da sich letztere auch auf nicht technische Systeme (z.B. Papier) bezieht.

Abgrenzungen zu Steganographie, Information Hiding, IT/Computer-Forensik, Watermarking

Security by Obscurity ist obsolet !!

Die Idee, dass die Sicherheit einer kryptographischen Methode höher ist, wenn man keine Details darüber preisgibt, ist ein Irrglaube !!

Warum ?

- Jede mögliche nach außen gedrungene Information über das verwendete Verfahren (z.B. durch MitarbeiterInnen die das Unternehmen frustriert verlassen) muss eine Änderung des Verfahrens nach sich ziehen, da sonst die Sicherheit gefährdet ist. Geheimhaltung ist nicht möglich !
- Es kann keine Standardisierung erfolgen und damit ist Interoperabilität und weitere Verbreitung unmöglich.
- Die Qualitätskontrolle einer breiten (kryptographischen) Öffentlichkeit fehlt notwendigerweise.

"Kerckhoff'sches Prinzip" - die Sicherheit eines Verschlüsselungsverfahrens liegt in der Geheimhaltung der verwendeten Schlüssel aber NICHT in der Geheimhaltung des verwendeten Verfahrens.

Beispiele für Verstösse gegen das Kerckhoff’sche Prinzip

- In Deutschland wurde vor Jahren die Verschlüsselung der Daten auf der Sozialversicherungskarte “geheim´´ durch den Bestbieter (eine kleine Firma) entwickelt. Die Karte war nur sehr kurze Zeit in Verwendung, ehe der Verschlüsselungalgorithmus (ein simples XOR Verfahren) geknackt wurde.
- Digital Rights Management - geheime CSS Verschlüsselung bei DVD geknackt und source re-engineered.
- Geheime A5 Verschlüsselung in GSM Kommunikation.
- PAY-TV (VideoCrypt) - hier wurde eine geheime Hash Funktion durch ehem. Mitarbeiter geleaked was zum Austausch von Tausenden Dekodern führte.

State-of-the-Art Verschlüsselung

"State of the art´´ sind also nicht geheime sondern allseits bekannte und überprüfte Algorithmen, die geheime Schlüssel, (Keys(K)) verwenden. Ein Key(K) kann jeden Wert aus einer großen Menge von Möglichkeiten annehmen (**Keyspace**). Die Sicherheit solcher Verfahren liegt also in der Geheimhaltung der Schlüssel und nicht des Verfahrens – diese genügen demnach dem "Kerckhoff'schen Prinzip".

$$E_K(M) = C \qquad D_K(C) = M$$

Manche Algorithmen benutzen unterschiedliche Schlüssel für das ver- und entschlüsseln.

$$E_{K_1}(M) = C \quad D_{K_2}(C) = M \qquad D_{K_2}(E_{K_1}(M)) = M$$

Solche Algorithmen können veröffentlicht und analysiert werden.

Ein **Kryptosystem** besteht aus Algorithmus $+$ Plaintext $+$ Ciphertext $+$ Keys.

Symmetrische Algorithmen

Bei diesen Algorithmen sind der Verschlüsselungskey und Entschlüsselungskey meist identisch, manchmal kann der Verschlüsselungskey aus dem Entschlüsselungskey berechnet werden und umgekehrt.

Problem: SenderIn und EmpfängerIn müssen sich auf einen gemeinsamen Key einigen. Dazu müssen sie sicher kommunizieren können (was ja eigentlich Ziel der Verschlüsselung ist und keine Vorraussetzung). Die Sicherheit dieses Systems liegt in der Sicherheit des Schlüssels, der geheim bleiben muß.

Grundsätzlich können zwei Kategorien unterschieden werden:

- **Stream Ciphers** arbeiten zu einem bestimmten Zeitpunkt *t* an einem einzelnen Bit (manchmal auch Byte).
- **Block Ciphers** arbeiten zu einem bestimmten Zeitpunkt *t* an einer Gruppe von Bits, der Blockgrösse. Typischerweise sind dies z.B. 128 Bits (wie bei AES).

Public-key Algorithmen

Diese Klasse von Algorithmen werden auch als asymmetrische Verfahren bezeichnet. Der Verschlüsselungskey und der Entschlüsselungskey sind nicht identisch. Der Entschlüsselungskey kann nicht (in vernünftiger Zeit) aus dem Verschlüsselungskey berechnet werden.

Public Key Der Verschlüsselungskey wird öffentlich bekannt gegeben und wird daher auch als Public Key bezeichnet. Jede Person, die Zugang zu diesem Schlüssel hat, kann eine Nachricht für den Eigner des Keys verschlüsseln. Eine Entschlüsselung ist damit aber nicht mehr möglich.

Private Key Der Entschlüsselungskey muß geheim gehalten werden und wird daher auch als Private Key bezeichnet. Jene Person, die ihn besitzt, kann eine mit dem zugehörigen Public Key verschlüsselte Nachricht entschlüsseln.

Public-Key Algorithmen haben den großen Vorteil, daß kein sicherer Kanal für die Übertragung eines gemeinsamen Schlüssels benötigt wird.

Ablauf Public-key Verschlüssleung

Schritte

1. Der Sender verschafft sich Zugang zum Public-Key des Empfängers (über einen potentiell unsicheren Kanal).
2. Der Sender verschlüsselt die Nachricht mit dem Public-Key des Empfängers und schickt sie diesem über den unsicheren Kanal.
3. Der Empfänger entschlüsselt die Nachricht mit seinem Private-Key.

Diese Verfahren werden auch für digitale Unterschrifen verwendet. In diesem Fall erfolgt die Verschlüsselung (d.h. die Unterschrift) mit dem Private Key, die Entschlüsselung (die Unterschriftsverifikation) mit dem Public Key.

Kryptoanalysis

In der Kryptographie wird angenommen dass ein Kryptoanalytiker (AngreiferIn, LauscherIn, FeindIn) kompletten Zugang zu den Kommunikationen zwischen Sender und Empfänger hat. Die zentrale Frage ist:
“Ist eine Erzeugung des Plaintextes ohne Kenntnis der Schlüssel möglich?´´
Attack, Attacke: eine versuchte Kryptoanalyse wird als Attack bezeichnet. Dabei wird davon ausgegangen, dass lt. Kerckhoff’schem Prinzip der/die FeindIn detaillierte Kenntnis des verwendeten Verfahrens hat.
Compromise: Ein Verlust des Schlüssels durch nicht-kryptoanalytische Methoden (Indiskretion, Social Engineering, ...) heißt Compromise. Dieser Ansatz ist oft am effizientesten.
Rubber-Hose Cryptoanalysis: Verwendung von Folter und Bedrohung um einen Key zu bekommen.

Attacken unterschieden sich typischerweise von durch verschiedene Fähigkeiten / zur Verfügung stehende Daten des Angreifers.

Ciphertext-only Attack: Der/die FeindIn besitzt den Ciphertext von verschiedenen Nachrichten, die mit dem gleichen Algorithmus verschlüsselt wurden.
geg: $C_1, \ldots C_i$
ges: $P_1, \ldots P_i$ oder Algorithmus um P_{i+1} aus C_{i+1}berechnen zu können.

Der einfachste Angriff dieses Typs ist das Durchprobieren aller Schlüssel, die sog. **brute force attack**. Fehlen die Resourcen um alle Kandidaten-Plaintexte zu untersuchen, kann man automatisiert diejenigen mit geringer Entropie heraussuchen (Ciphertext mit falschem Key entschlüsselt liefert Daten mit hoher Entropie). Ebenfalls in diese Kategorie zählen Angriffe die bekannte Symbol-Häufigheitsverteilungen ausnutzen, z.B. gegen den Caesar Cipher.

Known Plaintext Attack: Der/die FeindIn kennt (beliebige) Ciphertexte und Plaintexte.

geg: $P_1, C_1, \ldots P_i, C_i$

ges: Key oder Algorithmus um P_{i+1} aus C_{i+1} berechnen zu können.

Bsp1: Briefe enden häufig mit bestimmten Floskeln wie “Mit freundlichen Grüssen” oder “Hochachtungsvoll”. Kennt ein Angreifer nun Ciphertexte solcher Grussformeln, kann so ein Angriff durchgeführt werden.

Bsp2: Im WkII kannten die Alliierten die deutschen Nachrichten im Fall von abgworfenen Wasserbomben, welche Zeitpunkt und Ort enthielten. Für die known-Plaintext Attacke warfen die Alliierten Wasserbomben ab, und kannten natürlich Zeitpunkt und Ort (known-Plaintext). Nach dem Abfangen des entsprechenden Ciphertexts konnte eine known-Plaintext Attacke durchgeführt werden, um den “Tagesschlüssel” zu ermitteln.

Arten von Attacken III

Chosen Plaintext Attack: ähnlich wie die Known Plaintext Attack, nur kann der/die AngreiferIen die Plaintexte auswählen, die verschlüsselt werden (bevor er weiss welcher Ciphertext entschlüsselt werden soll). Dadurch können bestimmte Plaintexte verwendet werden, die mehr Information über die Schlüssel liefern. D.h. $P_1, \ldots P_i$ sind frei wählbar.

Adaptive-Chosen Plaintext Attack: Der/die AngreiferIn kann den Plaintext adaptiv verändern (nachdem er weiss welcher Ciphertext entschlüsselt werden soll), je nach Ergebnis des letzten Ver- und Entschlüsselungsprozesses.

Chosen Plaintext Angriffe können gegen Public-Key Verschlüsselung immer durchgeführt werden, da der Public-Key öffentlich verfügbar ist, und jeder beliebige Plaintext, zu jeder beliebigen Zeit verschlüsselt werden kann. Dies ist ein grundsätzlicher Nachteil von Public-Key Verschlüsselung, daher müssen die verwendeten Verfahren gegen alle Arten von Known Plaintext Attacken widerstandsfähig sein. Das ist bei symmetrischen Verfahren nicht so.

Arten von Attacken IV

Chosen-Ciphertext Attack: Der/die FeindIn kann verschiedene Ciphertexte auswählen, die entschlüsselt werden und hat Zugang zum entschlüsselten Plaintext. geg: C_1, P_1, $\ldots C_i$, P_i, wobei $C_1, \ldots C_i$ frei wählbar sind
ges: Key oder Algorithmus um P_{i+1} aus C_{i+1}berechnen zu können.

Diese Attacke ist z.B. realistisch wenn ein Public Key System verwendet wird um zu signieren: da der Verifikationskey öffentlich verfügbar ist, kann ein Ciphertext (signiert - verschlüsselt - mit dem private Key) jederzeit entschlüsselt werden.
Die “Lunchtime” oder “Midnight” Attack ist ein Modell für non-Adaptivität - der Angeifer hat z.B. die Mittagszeit zur Verfügung, um mit dem Rechner des Opfers Ciphertexte zu entschlüsseln, er kennt den zu entschlüsselnden Ciphertext noch nicht (CCA1 Angriff). Der adaptive Angriff (CCA2) erlaubt das Entschlüsseln beliebiger Ciphertexte auch nach Bekanntgabe des Zieltexts, mit Ausnahme des Zieltexts. Kaum praktische Attacken dieser Form.

Angriffsziele - wann ist ein Algorithmus gebrochen ?

1. **Total Break**: Der Schlüssel wird gefunden.
2. **Global Deduction**: Es wird ein alternativer Algorithmus zum Berechnen von $D_K(C)$ gefunden.
3. **Local Deduction**: Es wird der Plaintext aus dem Ciphertext einmalig ermittelt.
4. **Partial Local Deduction**: Es wird ein Teil des Plaintexts aus dem Ciphertext einmalig ermittelt.
5. **Information Deduction**: Eine Information über den Schlüssel oder den Plaintext wird gewommen.

Letzteres kann sich beispielsweise auf die Struktur des Schlüssels beziehen: wenn man weiss, dass der Schlüssel 65% aus 1 und nur 35% aus 0 besteht, muss in einer brute-force Attacke deutlich weniger gerechnet werden. So eine Erkenntnis kann z.B. durch eine "Timing Attacke" gewonnen werden, in der aus dem gemessenen Zeitverbrauch bei einer Entschlüsselung Schlüsselstruktur abgeleitet werden kann (je mehr 0en, desto schneller). Ähnliches ist durch Messung des Stromverbrauchs möglich.

Sicherheit von Verschlüsselungsalgorithmen

In praktischen Szenarien spielen noch andere Faktoren als die rein kryptographische Sicherheit eine Rolle:

- Wenn die Kosten einen Algorithmus zu brechen höher sind als der Wert der Daten, dann ist der Algorithmus *wahrscheinlich* sicher.
- Wenn die Zeit, einen Algorithmus zu brechen größer ist, als die Zeit, die die Daten geheim sein müssen, dann ist der Algorithmus *wahrscheinlich* sicher.

Warum *wahrscheinlich***:** Es gibt immer wieder unerwartete Durchbrüche in der Kryptoanalyse, z.B. das Brechen der Hashfunktionen SHA-1 und MD-5, das Brechen aller Knapsack-basierten Public-Key Algorithmen, ...

Bemerkung: in einer konkreten Anwendung wird man nicht den "sichersten" aller bekannten Algorithmen einsetzen, sondern einen, dessen Sicherheitsniveau der Aufgabenstellung angemessen ist (sichere Algorithmen verursachen typischerweise höhere Kosten); z.B. long time Archivierung vs. Absicherung eines Mobilfunkgesprächs; militärische Kommunikation vs. Messenger Dienst.

Sicherheitsbegriffe I

Es ist im Allgemeinen nicht sinnvoll, bei kryptographischen Verfahren von "Sicherheit" zu sprechen, ohne diesen Begriff genauer zu qualifizieren. Ein Sicherheitsbegriff besagt, dass ein Angreifer ein bestimmtes Ziel nicht erreichen kann. Die Definition besteht also aus zwei Teilen: Einem Angreifer mit genau beschriebenen Fähigkeiten und dem Sicherheitsziel.

Computational Security
: Ein Verschlüsselungsverfahren ist mit den zur Verfügung stehenden Mitteln (bester Algorithmus, Daten, Speicherbedarf, Komplexität - Rechenkapazität) nicht brechbar. Für kein praktisches Verschlüsselungsverfahren kann unter diesen Bedingungen Sicherheit bewiesen werden. Typischerweise wird so die Sicherheit bzgl. spezieller Attacken (z.B. brute force) analysiert, die dann aber nichts über die Sicherheit gegenüber anderen Attacken aussagt. Moore's Law und neue Rechenparadigmen wie Quantencomputer stellen hier ein Problem dar.

Sicherheitsbegriffe II

Provable Security : Ein anderer Ansatz um Evidenz von Sicherheit zu erlangen ist das Zurückführen der Sicherheit eines Algorithmus auf ein wohlbekanntes schwieriges Problem, z.B. “das Verschlüsselungsverfahren ist sicher wenn eine bestimmte Zahl nicht faktorisiert werden kann”. Hier handelt es sich um einen relativen Sicherheitsbegriff aber keinen Absoluten: im Fall von RSA ist klar dass verbesserte Faktorisierung die Sicherheit gefährdet, aber es ist nicht bewiesen dass das die einzige Möglichkeit ist RSA anzugreifen.

Unconditional Security : Es ist egal wieviel Information über den Ciphertext vorhanden ist und wieviel Rechenkapazität zur Verfügung steht: Es reicht nicht aus, um damit den Plaintext zu ermitteln.

Weiters betrachten wir Sicherheit im Sinn der Un-unterscheidbarkeit (z.B. IND-CPA, IND-CCA), perfekte Sicherheit und semantische Sicherheit.

Um über die Sicherheit von konkreten Verschlüsselungssystemen entscheiden zu können, muss eine formale Beschreibung der Angriffarten und der möglichen Erfolge eines Angriffs möglich sein. Bei dieser Analyse ist nicht die konkrete Situation z.B. bzgl. Geräteausstattung des Angreifers relevant, entscheidend ist vielmehr über welche Informationen in welchem Umfang er verfügt. Man simuliert die konkrete Situation durch den Einsatz eines sog. **Orakels**. Der Angreifer kann Anfragen an das Orakel stellen und erhält dem Angriffsszenario entsprechend bestimmte Informationen, wobei das Orakel das angegriffene Verschlüsselungssystem kennt. Ein Orakel ist damit ein formales und technisches Hilfsmittel, um Angriffe beschreiben zu können, unabhängig von konkreten Situationen. Erhält ein Angreifer vom Orakel die konkret zu lösende Aufgabe **nach** dem Angriff, spricht man von einem **direkten Angriff**. Wenn der Angreifer die zu lösende Aufgabe kennm bevor er den Angriff durchführt, und daher den Angriff gezielt auf die Aufgabe abstimmen kann, so heisst der Angriff **adaptiv**.

Angriffstypen im Orakel Modell

- *Ciphertext-only Attack*: das Orakel wählt zufällige Plaintexte, verschlüsselt diese und stellt die Ciphertexte dem Angreifer zur Verfügung.
- *Known Plaintext Attack*: das Orakel wählt zufällige Plaintexte, verschlüsselt diese und stellt die Plaintexte und Ciphertexte dem Angreifer zur Verfügung.
- *Chosen Plaintext Attack*: der Angreifer wählt Plaintexte und schickt diese an das Orakel, welches diese zu Ciphertexten verschlüsselt und an den Angreifer schickt.
- *Chosen Ciphertext Attack*: der Angreifer wählt Ciphertexte aus und schickt diese an das Orakel, welches diese zu Plaintexten entschlüsselt und an den Angreifer schickt.

Um den Erfolg eines Angreifers zu messen, stellt ihm das Orakel eine Aufgabe. Wenn er diese lösen kann, war der Angriff erfolgreich.

Angriffserfolg im Orakelmodell

Für die **Total Break** muss der Angreifer den vom Orakel genutzten (geheimen) Schlüssel bestimmen. Kann er das, ist der Angriff erfolgreich. Für die **partial local deduction** muss der Angreifer Teile des Plaintextes angeben können um erfolgreich zu sein. Wird der Angriff als chosen Ciphertext Attack ausgeführt, darf das Orakel nicht den “Aufgaben Ciphertext” entschlüsseln, nur andere, die der Angreifer wählt.

Wie läuft dies bei einem adaptive Chosen Plaintext Angriff ab:

1. Orakel wählt Schlüssel K und Plaintext P zufällig aus und schickt den Ciphertext C an den Angreifer,
2. Der Angreifer wählt selbst “interessante” andere Plaintexte, schickt diese an das Orakel und lässt diese verschlüsseln, und erhält die Ciphertexte zurück. Meist setzt man voraus, dass der Angreifer die Ciphertexte speichern können muss.
3. Kann der Angreifer C ermitteln, hat er die Aufgabe gelöst.

Angriffs Misserfolg im Orakelmodell

Bisher wurde der Fall noch nicht formalisiert, in dem der Angreifer einen Misserfolg hat, d.h. er gewinnt keine Information aus seinem Angriff. Hier ist allerdings zu beachten, dass ein Angreifer wesentlich mehr Informationen haben kann als in der formalen Beschreibung. Sind Sender und Empfänger bekannt, gibt es Vermutungen über den Inhalt der Nachrichten, d.h. der Angreifer kann versuchen den Plaintext zu raten.
Eine gute Verschlüsselungsfunktion kann in diesem Fall nur erreichen, dass der Angreifer anhand der Ciphertexte nicht entscheiden kann, ob er richtig geraten hat (d.h. ob er dem richtigen Plaintext nahe kommt durch Raten). Er kann also weder Ciphertexte entschlüsseln, noch kann er sie als Test benutzen, ob seine Rateversuche korrekter werden. Alle Rateversuche des Angreifers haben vor dem Angriff die gleiche Wahrscheinlichkeit korrekt zu sein, wie nach dem Angriff. Genau das bedeutet, dass der Angreifer keine Information gewonnen hat, er lernt nicht.

1. Ein Angreifer wählt zwei Plaintexte aus und schickt beide an das Orakel.
2. Das Orakel wählt einen Key aus dem Keyspace, wählt zufällig einen der zwei Plaintexte, verschlüsselt diesen mit dem Key und schickt den Ciphertext an den Angreifer.
3. Der Angreifer bestimmt, welcher der beiden Plaintexte vom Orakel verschlüsselt wurde.
4. Wenn der Angreifer den richtigen Plaintext nur mit Wahrscheinlichkeit $1/2$ - d.h. nicht wesentlich besser als - bestimmen kann (also raten muss), hat er durch den Angriff keine Information gewonnen.

Das so geprüfte Verschlüsselungsverfahren ist sicher im Sinne der Un-unterschiedbarkeit (polynomielle Un-unterscheidbarkeit: wenn der Angreifer mit einem Algorithmus polynomieller Komplexität nicht besser als $1/2$ ist). Wird auch als **IND-CPA** bezeichnet (indistinguishability under chosen plaintext attack), was als äquivalent zur semantischen Sicherheit gezeigt wurde (siehe später).

IND-CCA1 und IND-CCA2

In diesen Fällen ist das Orakel auch zur Entschlüsselung fähig – dies modelliert unterschiedliche Fähigkeiten eines Angreifers.

1. Der Angreifer schickt bel. Ciphertexte an das Orakel, erhält die entsprechenden Plaintexte, oder führt andere Berechnungen aus.
2. Der Angreifer wählt zwei Plaintexte aus und schickt an das Orakel.
3. Das Orakel wählt einen Key aus dem Keyspace, wählt zufällig einen der zwei Plaintexte, verschlüsselt diesen mit dem Key und schickt den Ciphertext C an den Angreifer.
4. Der Angreifer bestimmt (führt entsprechende Berechnungen durch), welcher der beiden Plaintexte vom Orakel verschlüsselt wurde.

IND-CCA1 Im nicht-adaptiven Fall kann der Angreifer dem Orakel keine Ciphertexte zur Entschlüsselung mehr schicken.

IND-CCA2 Im adaptiven Fall kann der Angreifer beliebige Ciphertexte entschlüsseln lassen, mit Ausnahme natürlich von C.

5. Wenn der Angreifer den richtigen Plaintext nur mit Wahrscheinlichkeit $1/2$ - d.h. nicht wesentlich besser als - bestimmen kann (also raten muss), hat er durch den Angriff keine

IND-CCA1 erlaubt wiederholte Interaktion in Schritt 1) und nimmt damit an dass die Sicherheit nicht mit der Zeit abnimmt. IND-CCA2 legt nahe dass der Aufruf an das Entschlüsselungsorakel dem Angreifer Vorteile bringen würde, wenn er C kennt (entsprechend massgeschneiderte Ciphertexte entschlüsseln lassen).

CPA : Bei symmetrischen Verfahren, bedeutet das den Sender zu manipulieren damit er gewünschte Wörter verwendet. Bei public-key Verfahren ist das trivial, da ja der public-key verfügbar ist.

CCA : Der Empfänger muss irgendwie dazu gebracht werden, dass ciphertexte entschlüsselt werden können. Bei assymetrischen Verfahren typischerweise durch Signaturverfahren, wo der Empfänger mit dem private Key verschlüsselt, und der Angreifer mit dem Public-key entschlüsseln kann (davon noch später).

Ein Verfahren das IND-CCA1 sicher ist, ist auch IND-CPA sicher. Ist ein Vefahren IND-CCA2 sicher, ist es IND-CPA & IND-CCA1 sicher.

IND-CPA und randomisierte Verschlüsselung

Werden bei einem symmetrischen Verschlüsselungsverfahren viele Nachrichten mit identischem Schlüssel verschlüsselt, ist ein CPA erfolgreich: ist bekannt, dass ein Ciphertext aus der Verschlüsselung eines bestimmten Plaintexts entsteht, kennt der Angreifer den Plaintext, sobald er diesen Ciphertext sieht. Bei public-key Verfahren ist das noch einfacher: durch die Verfügbarkeit des public-key kann ein Angreifer selbst die beiden Challenge Plaintexte verschlüsseln und sofort anhand der Ciphertexte den richtigen Plaintext im Orakelmodell identifizieren. Jedes **deterministische** Verschlüsselungsverfahren (d.h. ein Plaintext führt verschlüsselt mit demselben Key immer zum gleichen Ciphertext) hat dieses Problem und ist daher nicht IND-CPA sicher !

Randomisierte Verschlüsselungsverfahren verwenden zusätzlich zum Key einen zusätzlichen, zufälligen Parameter der die eindeutige / deterministische Beziehung zwischen Plaintext und Ciphertext bricht.

Beispiel der CPA auf deterministische Verschlüsselung

Ein Bankkunde handelt mit Aktien und sendet verschlüsselt (symmetrisch) eine von drei Anweisungen: “Kaufen”, “Halten” oder “Verkaufen”. Ein Angreifer braucht nur einmal zu erfahren dass der Kunde gekauft hat (d.h. “Kaufen” ist dann der known/chosen Plaintext), und kennt im Folgenden den entsprechenden Ciphertext. Damit kann er bei jedem Auftauchen von diesem Plaintext die Information ableiten dass der Kunde gekauft hat. Ist das Verschlüsselungsverfahren randomisiert, nützt es dem Angreifer nichts wenn er einmal den Ciphertext zu “Kaufen” sieht, weil dieser bei jeder Verschlüsselung anders ist.

Bei public-key Verfahrne ist das noch einfacher: der Angreifer verschlüsselt selbst die drei Anweisungen mit dem public-key (“Kaufen”, “Halten” oder “Verkaufen”) und kennt damit die drei Ciphertexte zu diesen drei Plaintexten. Beim Auftreten eines dieser Ciphertexte ist die Aktion des Kunden sofort klar.

Wie (und ob) die Randomisierung bei symmetrischen und public-key Verfahren umgesetzt wird, diskutieren wir bei den jeweiligen Verfahren.

Perfekte Sicherheit nach Shannon

Ausgangspunkt ist ein allmächtiger (d.h. bel. grosse Resourcen) Angreifer, der A und B belauscht, und den Inhalt einer verschlüsselten Nachricht bestimmen will, die A an B geschickt hat. Der Angreifer hat zwei Möglichkeiten:

1. Der Angreifer kann raten ohne den Ciphertext gesehen zu haben: z.B. ist es wahrschenlicher dass der Plaintext “Hallo B” enhält als “kxfrgls”. Jeder Plaintext hat (im Abhängigkeit vom Dateiformat oder Inhalt) eine gewisse **a priori Wahrscheinlichkeit**, die bereits vor dem Angriff feststeht und von diesem unabhängig ist.
2. Der Angreifer versucht nach dem Angriff (bei dem er den Ciphertext analysiert) Rückschlüsse auf den Plaintext zu ziehen. Damit haben Plaintexte auch eine bestimmte **a posteriori Wahrscheinlichkeit** mit der sie auftreten - bestimmte Plaintexte sind nach der Analyse wahrscheinlicher geworden, andere weniger wahrscheinlich.

Seien **P**, **C**, und **K** die Mengen der möglichen Plaintexte, Ciphertexte und Schlüssel eines Verschlüsselungsverfahrens. X, Y, K seien Zufallsvariablen und $P(X = x)$, $P(Y = y)$ und $P(K = k)$ sind die Wahrscheinlichkeiten, dass X, Y, K die Werte (x, y, k) aus **P**, **C**, und **K** annehmen.

$P(X = x, Y = y)$ ist die Wahrscheinlichkeit dass X den Wert x aus **P** annimmt und Y den Wert y aus **C**. $P(X = x|Y = y)$ ist die **bedingte** Wahrscheinlichkeit, dass dass X den Wert x aus **P** annimmt unter der Annahme dass Y den Wert y aus **C** hat.

Zwei auf **P** und **C** diskret verteilte Zufallsvariable heissen **unabhängig**, wenn $P(X = x, Y = y) = P(X = x) * P(Y = y)$ gilt für alle x aus **P** und y aus **C**.
Allgemein gilt: $P(X = x, Y = y) = P(X = x|Y = y) * P(Y = y)$.

Formalisierung der perfekten Sicherheit II

Bsp: Angenommen der Angreifer weiss, dass A und B mit einem Shift-Cipher arbeiten. Wenn der Angreifer “nohoh” abfängt, kann der Plaintext nicht “hallo” gewesen sein (weil es sonst im Ciphertext auch zwei identische Zeichen hintereinander geben müsste). Damit hat “hallo” die a posteriori Wahrscheinlichkeit 0.
Intuitiv ist ein Verschlüsselungsverfahren sicher, wenn die Analyse des Ciphertexts keinerlei Informationen über den Plaintext liefert (damit ist der Shift Cipher offenbar nicht sicher). Das Auftreten eines bestimmten Klartext muss nach dem erfolgten Angriff genauso wahrscheinlich sein, wie vor dem Angriff. Damit müssen die a priori and a posteriori Wahrscheinlichkeiten für Plaintexte gleich sein.
Ein Verschlüsselungssystem heisst **perfekt sicher**, wenn für alle x aus **P** und y aus **C** gilt:

$$P(X = x) = P(X = x|Y = y)$$

Das Auftreten eines bestimmten Ciphertexts y ändert nichts an der Auftrittswahrscheinlichkeit von x.

Berechnung der a posteriori Wahrscheinlichkeit I

Bei der Bewertung ob ein Verschlüsselungsverfahren perfekt sicher ist muss man nachweisen dass die a priori und die a posteriori Wahrscheinlichkeit für jeden Plaintext übereinstimmen. Die schwierige Aufgabe ist die Bestimmung der a posteriori Wahrscheinlichkeit.

Sei $P(K = k)$ die Wahrscheinlichkeit, dass A und B den Schlüssel k aus **K** gewählt haben. X und K sind unabhängig, weil der Schlüssel typischerweise nicht in Abhängigkeit vom Plaintext gewählt wird. **C**(k) ist die Menge aller Ciphertexte die mit diesem Schlüssel erzeugt werden kann: $\{E_k(x)|x \in \mathbf{P}\}$.

Die Wahrscheinlichkeit einen bestimmten Ciphertext zu erhalten wird bestimmt durch die Wahrscheinlichkeit dass man ein Schlüssel/Plaintext Paar (k, x) hat, sodass $y = E_k(x)$. Durch die Unabhängigkeit von Schlüssel und Plaintext erhält man:

$$P(Y = y) = \sum_{k|y \in \mathbf{C}(k)} P(K = k, X = D_k(y))$$

und damit $= \sum_{k|y \in \mathbf{C}(k)} P(K = k) * P(X = D_k(y))$.

Die bedingte Wahrscheinlichkeit dass y auftritt wenn man weiss dass der Plaintext x ist wird bestimmt als die Wahrscheinlichkeit der Schlüssel sodass $E_k(x) = y$:

$$P(Y = y|X = x) = \sum_{x \in D_k(y)} P(K = k) \; .$$

Wendet man nun die Formel von Bayes an (für $P(Y = y) > 0$)

$$P(X = x|Y = y) = \frac{P(X = x) * P(Y = y|X = x)}{P(Y = y)} \text{ so erhält man}$$

$$P(X = x|Y = y) = \frac{P(X = x) * \sum_{x \in D_k(y)} P(K = k)}{\sum_{k|y \in \mathbf{C}(k)} P(K = k) * P(X = D_k(y))} \; .$$

Diese Formel stellt die a posteriori Wahrscheinlichkeit durch meist leicht(er) bestimmbare Wahrscheinlichkeiten her.

Charakterisierung von perfekt sicheren Verschlüsselungssystemen (Shannon)

HÜ1: Zeigen sie durch die eben hergeleitete Formel, dass der Shift Cipher, wenn alle 26 Schlüssel z gleich wahrscheinlich sind, perfekt sicher ist - für einbuchstabige Plaintexte.

Charakterisierung perfekter Sicherheit: Seien **P**, **C**, und **K** die Mengen der möglichen Plaintexte, Ciphertexte und Schlüssel eines Verschlüsselungsverfahrens und E eine Abbildung von **P** nach **C**. Weiters ist $|\mathbf{P}| = |\mathbf{C}| = |\mathbf{K}|$ und $P(X = x) > 0$. Das Verschlüsselungsverfahren E ist genau dann perfekt sicher, wenn jeder Schlüssel mit gleicher Wahrscheinlichkeit gewählt wird und für alle $x \in \mathbf{P}$ und $y \in \mathbf{C}$ genau ein Schlüssel existiert mit $E_k(x) = y$.

Das bedeutet, dass es so viele Schlüssel geben muss wie Plaintexte. Insbesondere folgt daraus, dass die Schlüssel mindestens so lang sein müssen wie die Plaintexte, was bei grossen Plaintexten offensichtlich problematisch ist!

Weitere Sicherheitsbegriffe

Weitere Sicherheitsbegriffe mit unterschiedlichen Angreiferzielen:

Semantische Sicherheit: Jeder Angreifer A, der aus dem Ciphertext und dem public-key Angaben über den Plaintext machen kann, kann das nur unwesentlich besser als ein Angreifer B, der den Ciphertext nicht kennt (keine Ununterscheidbarkeit sondern Fähigkeit Vorhersagen über den Plaintext zu machen).
Nicht-Verformbarkeit: heisst auch *non-malleability (NM)* und bezeichnet Verfahren, bei denen es unmöglich ist, einen Ciphertext (systematisch) so zu verändern, dass der entsprechende Plaintext zum gesuchten Plaintext in einem bestimmten Verhältnis steht.
Real or Random Sicherheit (ROR): ein Angreifer kann nicht unterscheiden, ob ein Ciphertext einen von ihm verschlüsselten Plaintext oder eine zufälligen verschlüsselten Plaintext darstellt.

NM-CCA2 ist äquivalent zu IND-CCA2 und ROR-CCA2, NM-CPA ist ein stärkerer Begriff als IND-CPA. Weiters ist perfekte Sicherheit gleichbedeutend mit IND-CCA.

Outline

1 Formalia

2 Einleitung
- Politik und Geschichte
- Terminologie
- Grundlagen: Einfache Methoden, Hash Funktionen, Protokolle

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit

Substitution Ciphers

Simple Substitution Ciphers ersetzen ein Symbol im Plaintext (häufig Character) durch ein entsprechendes Symbol im Ciphertext nach einer Regel oder Tabelle (Shift- oder Caesar Cipher ist ein einfaches Bsp.), heissen auch *Monoalphabetic Ciphers*.

Cryptoanalyse durch Ciphertext-only Attacke verwendet eine Häufigkeitsanalyse der verwendeten Symbole, z.B. sind die Auftrittswahrscheinlichkeiten von einzelnen Buchstaben oder Buchstaben Tupeln und Tripeln für alle Sprachen bekannt. Durch einen Abgleich Häufigkeitshistogramme zwischen Plain- und Ciphertext wird die Substitution erkannt.

Keine Resistenz gegen known-Plaintext (oder stärkere) Angriffe bei Einzelanwendung, gilt ebenso für die folgenden Varianten !

Homophonic Ciphers Ein Symbol im Plaintext hat mehrere Möglichkeiten der Abbildung in den Ciphertext, z.B. Anzahl der Abbildungen in Abhängigkeit von der Häufigkeit des Symbols (um Häufigkeitsanalysen zu verunmöglichen). Welche Abbildung gewählt wird kann vom Kontext im Plaintext abhängen oder zufällig gewählt werden – Verwendung im amerikanischen Bürgerkrieg, Textverarbeitung Word Perfect.

Polygram Ciphers Blöcke im Plaintext werden durch Blöcke im Ciphertext ersetzt (Playfair Cipher in WK1).

Polyalphabetic Ciphers sind Zusammensetzungen von mehreren Substitution Ciphers, deren Verwendung z.B. von der Position im Key abhängt. *Rotor Maschinen* wie die Enigma realisieren das in Hardware. Ein Spezialfall ist der sog. Vigenère Cipher, der aus mehreren monoalphabetischen Ciphers des Ceasar (Shift) Ciphers besteht. Eigentlich entwickelt von Giovan Battista Bellaso 1553, 1863 von Kasiski gebrochen (Ciphertext-only Angriff).

Die Symbole des Plaintextes bleiben gleich. Es wird nur deren Anordnung geändert, z.B. die Zeilen werden als Spalten angeordnet oder nach einer Vorschrift permutiert. Solche Verfahren wurden auch bei der Verschlüsselung von Fernsehkanälen im Bereich hybrid Pay-TV verwendet.

Da die Häufigkeitsverteilung der Symbole gleich bleibt, wie im ursprünglichen Plaintext, bietet deren Verteilung auch keinen verwertbaren Hinweis auf die angewandte Transposition / Permutation.

Transposition (oder Permutation) kann einer known-Plaintext Attacke (oder stärkerer Attacke) nicht widerstehen.

XOR Verschlüsselung

XOR wird auch als ausschließendes Oder bezeichnet. Das mathematische Symbol ist $\oplus$ bzw. ^ in der Programmiersprache C.

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Symmetrische Algorithmen mit Schlüssel:

$$P \oplus K = C$$

$$C \oplus K = P$$

weil $a \oplus b \oplus b = a$, und damit $C \oplus K = P \oplus K \oplus K = P$.

Short-Key XOR als Polyalphabetischer Substitution Cipher

Short-Key XOR verwendet einen binären Key bestimmter Länge, der durch wiederholtes Padding auf die Länge des Plaintexts gebracht wird. Wurde historisch zum Teil als “almost as secure as DES´´ bezeichnet, NSA erlaubte in den USA den amerikanischen Telefongesellschaften das Benutzen dieses Systems (verdächtig !).

Wesentlich für die Sicherheit ist die Länge des Keys – im Falle von wiederholt angewendeten kurzen Keys (d.h. kürzer als der Plaintext lang ist) ist das Sytem allerdings leicht zu knacken.

Im Folgenden nehmen wir an, dass wir einen Text natürlicher Sprache als Plaintext haben (und die ASCII character binär kodiert sind). Somit ist XOR direkt anwendbar, der Key besteht ebenfalls aus binär codierten (mehreren) Characters. Das anschliessend beschriebene Kryptoanalysis Verfahren ist nicht auf Text beschränkt, setzt allerdings ein Symbol Alphabeth mit Redundanz voraus (d.h. Elemente das Alphabeths, z.B. Bytes, treten nicht gleich häufig auf).

1. Die Key length kann durch counting coincidences bestimmt werden. Dabei wird ein XOR des Ciphertextes auf geshiftete (verschobene) Varianten des Ciphertextes angewendet und die gleich bleibenden Bytes (ein Byte kodiert einen Character) gezählt.
   Mehrfaches der Key length: 6.65% gleichbleibende Bytes
   Sonstige Positionen: 3.85% gleichbleibende Bytes
   $\Longrightarrow$ Index of coincidence: Der kleinste Shift mit 6.65% ergibt die Keylength.
2. Wenn der Ciphertextes um die Größe der Keylength verschoben und mit dem unverschobenen Text ein XOR angewendet wird, wird der Key entfernt (XOR Rechenregeln ! s.u.). Es bleibt dann der Plaintext übrig, auf den mit dem verschobenen Plaintext ein XOR angewendet wurde. Der Plaintext ist dann einfach zu rekonstruieren.

Wie ist das einzusehen ?

- Jedes Character im Key definiert einen eigenen monoalphabetischen Substitution Cipher. Grundsätzlich beruht der Angriff auf dem sog. “Index of coincidence” (IoC), der die Wahrscheinlichkeit angibt, dass beim zufälligen Ziehen von zwei Characters aus einem Text diese identisch sind. Handelt es sich um ein Alphabeth mit 26 Elementen und einen verschlüsselten Text, so sollte die Wahrcheinlichkeit bei $1/26 = 3,85\%$ liegen. Englischsprachige Plaintexte haben einen IoC von $6,65\%$ (da z.B. “E” sehr häufig vorkommt).
- Wesentlich ist, dass monoalphabethische Substitution Ciphers den Index of coincidence nicht verändern (das ist ja auch genau die Idee der Häufigkeitsanalyse). Polyalphabethische Ciphers jedoch verändern den IOC in Richtung $3,85\%$.

Wie ist das einzusehen ?

- Betrachten wir nun einen Ciphertext ge-XORed mit einer geshifteten Version seiner selbst. Ist die Grösse des Shifts unabhängig von der Länge des Keys, wird jeder Character mit einem Character eines unterschiedlichen Alphabeths ge-XORed, was praktisch zufällig ist und eine 0 (=Übereinstimmung der Character lt. XOR) wird wieder in $3,85\%$ der Positionen auftreten.
- Ist jedoch der Shift ein Vielfaches der Key-länge, wird jeder Character mit einem Character des gleichen Alphabeths (des gleichen monoalphabetischen Ciphers) ge-XORed. Es ist daher zu erwarten, dass die 0 bei XOR so oft auftritt, wie zwei zufällige Plaintext-characters übereinstimmen, also bei $6,65\%$ aller Fälle.

Wie ist das einzusehen ?

- Formal: seien p_1 und p_2 plaintext sowie k_1 und k_2 key characters:

$$c_1 = p_1 \oplus k_1 \text{ sowie } c_2 = p_2 \oplus k_2 \,.$$

  Werden nun die beiden Ciphertext characters ge-XORed erhält man:

$$c_1 \oplus c_2 = p_1 \oplus k_1 \oplus p_2 \oplus k_2 = p_1 \oplus p_2 \oplus k_1 \oplus k_2 \,.$$

- Sind nun die beiden Ciphertext Characters ein Vielfaches der Key-länge voneinander entfernt, gilt $k_1 = k_2$ und damit $k_1 \oplus k_2 = 0$. Dieser Wert ist (1) Key-unabhängig (d.h. der Key wurde entfernt) und (2) entspricht offensichtlich dem Vorgehen zur Bestimmung des IoC im Plaintext (zwei Plaintext Character werden auf Gleichheit verglichen).

Alternative Methoden zur Bestimmung der Key-length

- *Methode von Kasiski*: Identische sich wiederholdende Teile des Plaintexts ge-XORed mit dem gleichen Teil des Keys ergeben sich wiederholende identische Ciphertext Teile. Die Grösse des Shifts zwischen dem Beginn solcher sich wiederholenden Teile im Ciphertext sollte daher ein Vielfaches der Key-Länge sein. Die Analyse der gemeinsamen Faktoren dieser Shifts identifiziert den häufigsten Faktor der dann der Key-länge entspricht.
- Alternativ lässt sich eine Iteration über die Länge der Keys durchführen, bei der die Hamming Distanz zwischen benachbarten Ciphertextblöcken der jeweiligen Keylänge berechnet wird (d.h. XOR und Zählen der 1en). Die Distanz muss natürlich auf die Key-länge normiert werden und der kleinste Wert liefert die Key-länge (wieder beruhend auf der Idee dass die Ciphertext Character die bei Shift um Key-länge auftreten aus dem gleichen Alphabeth stammen und daher den höheren IoC haben, was zu kleiner Hamming Distanz führt).

Entschlüsselung bei bekannter Key-length

1. Es liegt offenbar ein polyalphabetischer Cipher mit L Alphabethen vor, es gibt also L entsprechende monoaplphabetische Cipher. Werden die Ciphertext Character jedes dieser Cipher zusammengruppiert lässt sich auf jeden dieser L Cipher eine (triviale) Häufigkeitsanalyse durchführen.
2. Wir verwenden den Plaintext ge-XORed mit einer geshifteten Variante der Plaintexts. Dies entspricht einem sog. “Text Autokey Cipher”, bei dem der Key für die XOR Berechnung aus dem Plaintext selbst abgeleitet wird. Klassischerweise wird bei diesem Cipher vor dem Plaintext noch ein Key unbekannter Länge eingefügt, bevor dieser mit dem Plaintext ge-XORed wird. In unserem Fall ist dieser Key bekannt (der Shift um L Character). Werden nun die Daten mit einem Key ge-XORed (häufige N-gramme, Silben) und der gewonnene Plaintext ist um L Character in die andere Richtung verschoben identisch, so ist dieser Key korrekt ($P \oplus K \oplus K = P$; erstes K hier ja ebenfalls P und zweites K gewählter Key).

Wird nicht ein kurzer repetitiver Schlüssel verwendet sondern einer der die gleiche Länge aufweist wie der Plaintext, spricht man von OTP Verschlüsselung, dabei $C = P \oplus OTP$. Wird jeder Schlüssel nur einmal verwendet und hat identische Auftrittswahrscheinlichkeit, so wurde das 1926 so-genannte Vernam Verfahren von C. Shannon 1949 als **perfekt sicher** bewiesen. Intuitiv macht ein zufälliger Key aus einem nicht-zufälligen Plaintext einen zufälligen Ciphertext.

Mögliche Angriffe: (1) gegen zufälliges Erzeugen des Keys (Zufallszahlengeneratoren) und (2) bei mehrfacher Verwendung eines Keys.

Probleme: nicht praxistauglich (symmetrisches Verfahren, damit Key-Management Problem), nur für sehr kurze Nachrichten geeignet (z.B. “Rotes Telefon´´ USA - Moskau).

Attacke bei mehrfacher Verwendung des OTP Keys

Seien gegeben zwei Ciphertexte $C_1 = P_1 \oplus K$ und $C_2 = P_2 \oplus K$. Eve hat C_1 und C_2 abgefangen und rechnet $C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K$. Durch die Kommutativität der XOR-Operation ergibt sich $P_1 \oplus P_2 \oplus K \oplus K$ und wegen $a \oplus a = 0$ bleibt $P_1 \oplus P_2$. Daraus kann Eve wieder durch Methoden der Häufigkeitsanalyse P_1 und P_2 isolieren.

Ist zusätzlich ein dazugehöriger Plaintext verfügbar (z.B. P_2, also eine known-Plaintext Attacke), kann durch Berechnung von $C_1 \oplus C_2 \oplus P_2 = P_1 \oplus P_2 \oplus P_2$ der andere Plaintext P_1 direkt berechnet werden.

Visualisierung des Angriffs I

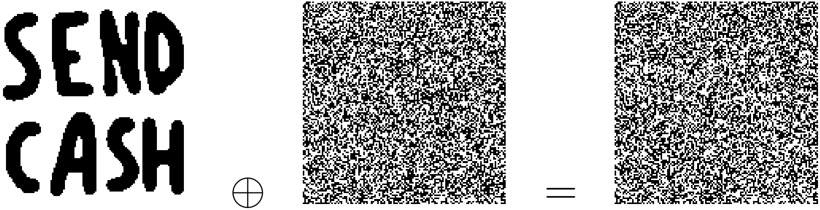


Figure: Binärbild wird mit OTP ge-XORed, ergibt zufälligen Cihertext.

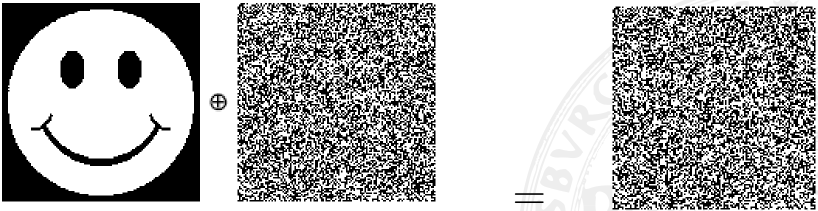


Figure: Ein anderes Binärbild wird mit dem gleichen OTP ge-XORed und ergibt ebenso zufälligen Ciphertext.

Visualisierung des Angriffs II

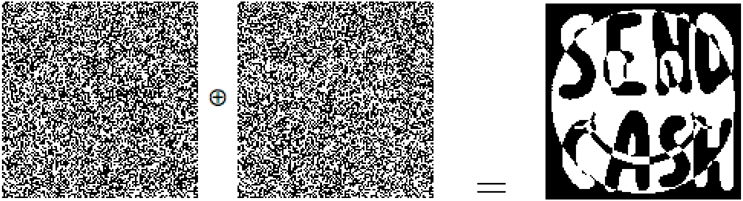


Figure: Werden nun die beiden Ciphertexte ge-XORed, ergibt sich ein XOR (Overlay) der beiden binären Plaintexte.

Figure: Bei Verfügbarkeit von einem zusätzlichen Plaintext wird dieser zum Overlay ge-XORed und ergibt den anderen Plaintext.

Kryptographische Protokolle

Kryptographie soll zum Lösen von Problemen verwendet werden. Sind nur die kryptographischen Algorithmen bekannt, bleibt es eine rein akademische Disziplin. Zur Verwendung im “Alltag´´ müssen weitere Vereinbarungen getroffen werden. Diese Abmachungen werden Protokolle genannt.

Ein **Protokoll** ist ein Ablauf von Vorgängen zwischen zwei oder mehreren Parteilen, um ein kryptographische Aufgabe zu erledigen. Dabei ist die Reihenfolge der Vorgänge meist von entscheidender Bedeutung.

- Jede/r TeilnehmerIn muß alle Schritte kennen.
- Jede/r muß zustimmen den Schritten zu folgen.
- Ein Protokoll ist eindeutig (nächster durchzuführender Schritt ist wohldefiniert).
- Für jede Situation gibt es einen definierten Schritt.

In einem Protokoll gibt es **TeilnehmerInnen** mit festgelegten Rollen:

Alice	initiiert alle Protokolle.
Bob	antwortet Alice.
Carol, Dave	sind weitere TeilnehmerInnen.
Eve (von Eavesdropper)	ist die passive Angreiferin bzw. Lauscherin.
Mallory (von Malicious, Malheur)	ist ein(e) aktive(r) AngreiferIn.
Trent	ist der Vertrauensmann der an der Sache desinteressiert Protokolle vervollständigt. Alle Parteien akzeptieren seine Aktivitäten als korrekt, (z.B. Notar oder Treuhänder, etwa bei einer Bezahlung mit Scheck, der prüft, ob der Scheck o.k. ist und bezahlt erst bei Erhalt der Waren.)
Walter, Peggy, Victor	je nach Bedarf.

Kryptographische Protokolle: Arten

- *Arbitrated Protocol* Ein Vertrauensmann (Arbitrator) wickelt so ein Protokoll ab - typischerweise kommerzielle Anbieter, ist immer ein Kommunkationsbottleneck, Frage nach Finanzierung !
- *Adjudicated Protocol* Protokoll mit zweiter Ebene, wird nur invoked wenn ein Problem in der ersten Ebene auftritt, nach dem Motto "nicht Betrug verhindern sondern aufdecken".
- *Self Enforcing Protocol* Das Protokoll selbst garantiert die Fairness.

Offensichtlich sind Protokolle des letzteren Typs die wünschenswerten, sind allerdings immer komplexer als die anderen Typen.

Kryptographische Protokolle: Angriffe

- *Passive Attack* Eve kann das Protokoll mithören. Entspricht einer Ciphertext-only attack.
- *Active Attack* Mallory verändert das Protokoll zu seinem Vorteil: täuscht vor, jemand anderes zu sein, erfindet neue Nachrichten im Protokoll, löscht, ersetzt, usw ...
- *Passive Cheating* Die AkteurInnen dieser Attackie sind diverse ProtokollteilnehmerInnen, die zwar dem Protokoll folgen, aber mehr Information wollen (und durch Lauschen diese auch bekommen), als ihnen zusteht.
- *Active Cheating* Die AkteurInnen dieser Attackie sind diverse ProtokollteilnehmerInnen, die während der Teilnahme das Protokoll zu ihrem Vorteil ändern (siehe active attack).

Kryptographische Protokolle: Bsp. Kommunikation mit symmetrischer Kryptographie

1. Alice und Bob einigen sich auf ein bestimmtes Kryptographiesystem.
2. Alice und Bob einigen sich auf einen gemeinsamen Key.
3. Alice verschlüsselt ihren Plaintext und erzeugt dadurch einen Ciphertext.
4. Alice schickt den Ciphertext an Bob.
5. Bob entschlüsselt den erhaltenen Ciphertext mit dem Key und erhält den Plaintext.

Eine Ciphertext-only Attacke passiert in Schritt 4; Alice könnte ihren Schlüssel an Eve weitergeben (und damit den Angriff trivialisieren), Bob und Alice müssen sich daher vertrauen.

Die Schlüsselverteilung muss über einen sicheren Kanal durchgeführt werden (schweirig bei grossen Systemen), wird in einem Netzwerk für jedes AnwenderInnen Paar ein verschiedener Key verwendet, ergeben sich $\frac{n(n-1)}{2}$ Keys, mit $n \ldots \#$ TeilnehmerInnen. Das ist natürlich nur bei einer kleinen Zahl von AnwenderInnen praktikabel.

Kryptographische Protokolle: Bsp. Kommunikation mit asymmetrischer Kryptographie

1. Alice und Bob einigen sich auf ein Public Key Kryptographiesystem
2. Bob “schickt” Alice seinen Public Key.
3. Alice verschlüsselt ihre Nachricht mit Bobs Public Key und schickt sie Bob.
4. Bob entschlüsselt die Nachricht mit seinem Private Key.

Viel einfacheres Schlüsselmanagement - der Public Key liegt in einer für alle zugänglichen Datenbank. Beim Auslesen des Public key muß garantiert sein, daß ein in der Datenbank deponierter Public Key auch wirklich von der richtigen Person stammt: Public key Authentifizierung, je nach geographischer Verteilung der TeilnehmerInnen verschiedene Methoden, z.B. Web of Trust, Digitale Signaturen und Zertifikate, Notariate,.... Bei *n* KommunikationsteilnehmerInnen werden (nur) *n* Schlüsselpaare benötigt.

Public Key Systeme sind KEIN Ersatz für symmetrische Algorithmen weil:

- Public Key Systeme sind langsamer (etwa um den Faktor 1000).
- Sie sind durch die Public key Verfügbarkeit IMMER durch Chosen-Plaintext Attacken angreifbar: $C = E(P)$, wobei P ein Plaintext aus einer Menge von n möglichen Plaintexten ist. Eine Kryptoanalyse muß nur alle n Plaintexte verschlüsseln und das Resultat mit C vergleichen, um P zu bestimmen.

Daher wird in den meisten praktischen Einsatzbereichen Public Key Kryptographie verwendet, um temporäre Keys (damit wird deren Verlust weniger kritisch) eines symmetrischen Verfahrens verschlüsselt zu verteilen.

Kryptographische Protokolle: Bsp. Kommunikation mit hybriden Systemen

1. Bob “schickt” Alice seinen Public Key.
2. Alice erzeugt einen zufälligen Session Key K, verschlüsselt diesen mit Bobs Public Key und schickt in zu Bob.

$$E_B(K)$$

3. Bob entschlüsselt Alices Nachricht mit seinem Private Key und erhält den Session Key.

$$D_B(E_B(K)) = K$$

4. Beide ver- und entschlüsseln ihre symmetrische Kommunikation mit dem Session Key K.

Bei diesem Verfahren wird also ein Session Key erzeugt und nach der Verwendung wieder zerstört. Das verringert die Gefahr im Fall von Schlüsselverlust. Aber auch fixe pre-installed keys können auf diese Weise ausgetauscht werden.

One-Way Functions

One-Way Functions sind eine wesentliche algorithmische Komponente der Kryptographie und sind Funktionen mit folgenden Eigenschaften:

- Bei gegebenem x ist es leicht $f(x)$ zu berechnen.
- Bei gegebenem $f(x)$ ist es sehr aufwändig (computationally infeasible bis unmöglich) x zu berechnen (=entschlüsseln).

Dieses Verfahren hat aber folgendes **Problem**: Nicht nur eine Attacke sondern auch das autorisierte “Entschlüsseln” wird damit erschwert bzw. unmöglich gemacht.
Daher gibt es *Trapdoor One-Way Functions*: mit den bereits beschriebenen Eigenschaften und zusätzlich mit einem “Geheimnis” Y. Wer immer Y kennt, kann x schnell aus $f(x)$ berechnen.

Bemerkung: Y wird im Fall von Public key Verfahren als private key bezeichnet, die Anwendung der one-way function ist die Verschlüsselung mt dem public key.

One-Way Hash Functions

Als Input wird ein pre-image M mit variabler Länge benutzt (die zu hashenden Daten), die Funktion H berechnet einen Output mit fixer Länge $h = H(M)$ (hash value). Kompressions- und Kontraktions Funktionen werden für die Realisierung der Eigenschaften verwendet. Das Ziel des Verfahrens ist ein (eindeutiger) "Fingerabdruck" h des pre-images M.

Durch die starke Kompression ist der Wertebereich des pre-image und des Hashwerts sehr unterschiedlich, d.h. es werden notwendigerweise unterschiedliche pre-images auf den gleichen Hashwert abgebildet. Damit erkennt das Verfahren erkennt aus einem Hash nicht notwendigerweise völlig eindeutig, ob zwei pre-images gleich sind.

Ein **Message Authentication Code** (MAC) ist eine One-Way Hash Funktion mit einem geheimen Key. Hash-value ist Funktion des pre-image und des Key, im einfachsten Fall wird der Hashwert symmetrisch verschlüsselt. Im Gegensatz zur Digitalen Signatur (Variante mit public-key) ist der MAC wesentlich schneller.

Digitale Unterschriften

Warum sind klassische - analoge - Unterschriften so wichtig?

1. Eine Unterschrift ist authentisch und identifiziert die/den Unterzeichnende/n (Unterschreiber ist wer er vorgibt zu sein),
2. nicht fälschbar (nicht von jemand anderem imitierbar),
3. nicht wiederverwendbar, z.B. auf einem anderen Dokument.
4. Das unterschriebene Dokument kann nicht verändert werden.
5. Die/der Unterschreibende kann die Unterschrift nicht leugnen.

Diese Aufzählung entspricht natürlich etwas mehr dem Wunschdenken als der Wirklichkeit. Trotzdem kommt der Unterschrift große Bedeutung zu. Bei einer digitalen Variante (“digitalisierte Unterschrift”, Unterschrift als .gif Bild) existieren weitere Probleme:

- (Unterschrifts)Dateien können kopiert werden.
- Unterschriften können mit Cut und Paste auf ein neues Dokument “geklebt´´ werden.
- Dateien können leicht modifiziert werden, ohne daß die Unterschrift beeinträchtigt wird.

Digitale Signatur: symmetrisch mit Arbitrator

Alice schickt Nachrichten an Bob. Trent kann mit Alice und Bob kommunizieren. Er teilt den geheimen Key K_A mit Alice und K_B mit Bob.

- Alice verschlüsselt ihre Nachricht an Bob mit K_A und schickt sie an Trent.
- Trent entschlüsselt mit K_A und verschlüsselt mit K_B.
- Trent schickt die Nachricht Bob.
- Bob entschlüsselt mit K_B und liest die Nachricht mit der Bestätigung, daß sie von Alice ist.

Nachteil: K_A und K_B müssen vereinbart werden, alles läuft über Trent (Bottleneck und Zahlungsfrage).

Digitale Signatur: asymmetrisch mit Hashfunktion

1. Alice erzeugt den One-Way Hash des zu unterschreibenden Dokumentes.
2. Alice verschlüsselt den Hash mit ihrem Private Key (=Unterschrift).
3. Alice schickt das Dokument und den Unterschriftshash an Bob.
4. Bob produziert One-Way Hash des Dokumentes und entschlüsselt den unterschriebenen One-Way Hash mit dem Public key von Alice. Stimmen die beiden überein, ist die Unterschrift gültig (d.h. Integrität des Dokuments gezeigt durch Hash Übereinstimmung und Authentizität des Unterschriftenleisters durch Verwendung des entsprechenden Public keys).

Folgende Bezeichnungen sind üblich: $S_K(m)$ Unterschrift mit Private Key ("sign"), $V_K(m)$ überprüfen der Unterschrift ("verify"). Die Verwendung der Hashfunktion ist **wesentlich**, da asymmetrische Verschlüsselungsverfahren zu langsam sind, grössere Datenmengen zu signieren (i.e. verschlüsseln).

One-Way Hash Functions - Eigenschaften

1. Bei gegebenem M soll h leicht zu berechnen sein (Geschwindigkeit).
2. Bei gegebenem h soll es schwierig sein M zu berechnen, sodaß $H(M) = h$ ist (first pre-image attack).
3. Bei gegebenem M soll es schwierig sein, eine andere Nachricht M' zu finden, sodaß $H(M) = H(M')$ ist (second pre-image attack).

Wenn Mallory die Punkte (2) und (3) durchführen kann, ist H nicht genügend sicher und Betrug leicht. Wenn, z.B. Alice eine Nachricht M digital unterschrieben hat, indem sie die Hash Funktion H auf M anwendet: $H(M)$, könnte Mallory M' erzeugen mit $H(M) = H(M')$. Somit könnte Mallory behaupten, Alice hätte ihm M' geschickt, was natürlich nicht der Wahrheit entspricht.

Eine Hash-Funktion H ist **Collision resistent**, wenn es schwierig ist zwei zufällige Nachrichten M und M' zu finden, sodaß $H(M) = H(M')$ ist.
Diese Eigenschaft ist wichtig bei der sogenannten “Geburtstags Attacke”, die auf folgendem kombinatorischen “Paradoxon” beruht:

1. “Wie viele Menschen müssen in einem Raum sein, damit die Wahrscheinlichkeit größer $\frac{1}{2}$ ist, das eine der Personen an einem vorgegebenen Tag Geburtstag hat?” (183)
2. “Wie viele Menschen müssen in einem Raum sein, damit die Wahrscheinlichkeit größer $\frac{1}{2}$ ist, das zwei Personen am gleichen Tag Geburtstag haben?” (23)

HÜ9: Leiten sie die kombinatorischen Werte des obigen Paradoxons her !

One-Way Hash Functions - Collision Resistance II

Diese beiden kombinatorischen Fragestellungen sind analog zu den folgenden:

1. Gegeben ist $H(M)$, suche M', sodaß $H(M) = H(M')$ (*der Hashwert ist fix, ich suche eine zweite Nachricht mit ebendiesem Hashwert - Geburtstag ist fix, suche Person die genau an diesem Tag Geburtstag hat*).
2. Suche M und M', sodaß $H(M) = H(M')$ (*der Hashwert ist beliebig, suche nur zwei Nachrichten die auf den gleichen hashwert abbilden - Geburtstag ist egal, suche nur zwei Personen die am gleichen Tag Geburtstag haben*).

Eine Hash Funktion produziert einen m-Bit Output. Um eine Nachricht zu finden, die auf einen gegebenen Wert abgebildet wird, müssen 2^m produziert werden. Um zwei Nachrichten zu finden, die den selben Hash-Wert liefern, müssen nur $2^{\frac{m}{2}}$ Nachrichen produziert werden.
HÜ10: Leiten sie die angegebenen Mengen von Hash-Werten her, die für die beiden Angriffsvarianten notwendig sind (2^m vs. $2^{m/2}$).

One-Way Hash Functions - Geburtstagsattacke

Das folgende Beispiel zeigt, wie diese Schwäche – eine nicht kollision-resistent Hash Funktion – ausgenützt werden kann.

Geburtstagsattacke

1. Alice besitzt zwei Versionen eines Vertrages, eine Version ist die echte, die andere wurde zu Ungunsten von Bob verändert.
2. Alice verändert beide geringfügig und berechnet die Hash-Werte der veränderten Dokumente (2^{32} verschiedene Dokumente bei 64-Bit Hash).
3. Alice sucht ein Paar von Hash-Werten, die gleich sind, und wählt dieses Dokumentenpaar für den Angriff.
4. Alice und Bob unterschreiben den Vertrag, der für Bob in Ordnung ist mit einem Protokoll, bei dem er den Hash des Dokumentes unterschreibt.
5. Irgendwann ersetzt Alice die Originalversion mit der anderen - equivalent unterschrieben von Bob !!

Offensichtlich spielt die Länge eines Hashwerts eine wesentliche Rolle bei der Beurteilung der Sicherheit (allerdings nicht nur !). Aus heutiger Sicht sind 64 und 128 Bit Hashlänge bereits deutlich zu kurz.
Der folgende Algorithmus erhöht die Länge des Outputs. Die Sicherheit dieser Variante ist allerdings nicht ganz klar, das Hash konkatinieren wird jedenfalls auch bei Keccak verwnedet (siehe dort).

1. Hash erzeugen
2. Hash an Nachricht anfügen
3. Hash der verlängerten Nachricht erzeugen.
4. Hash anhängen und neuen Hash erzeugen, kann beliebig oft wiederholt werden.

Der Input (die Nachricht M) und der Output der Funktion werden also auf vorhergehende Nachrichtenteile angewendet.

$$h_i = f(M_i, h_{i-1})$$

Digitale Unterschriften mit Verschlüssselung

1. Alice unterschreibt m mit ihrem Private Key $S_A(m)$.
2. Alice verschlüsselt unerschriebenes m mit Bobs Public Key und schickt ihm $E_B(S_A(m))$.
3. Bob entschlüsselt m mit seinem Private Key $D_B(E_B(S_A(m))) = S_A(m)$.
4. Bob überprüft die Unterschrift von Alice mit ihrem Public Key $V_A(S_A(m)) = m$.

Somit ergibt sich also eine Unterschrift sowohl auf dem Umschlag als auch auf dem Brief. Es werden für das Verschlüsseln und Unterschreiben verschieden Schlüsselpaare verwendet (außerdem: timestamps).

Digitale Empfangsbestätigungen

Immer, wenn Bob eine Nachricht empfangen hat, schickt er sie als Bestätigung signiert und verschlüsselt wieder zurück, sodass der Absender verifizieren kann, dass Bob die intendierte Nachricht erhalten hat.

1. Alice unterschreibt, verschlüsselt und schickt $E_B(S_A(m))$.
2. Bob entschlüsselt und überprüft die Unterschrift $V_A(D_B(E_B(S_A(m)))) = m$.
3. Bob unterschreibt m mit seinem Private Key, verschlüsselt dann mit Alices Public Key und schickt das ganze an Alice zurück $E_A(S_B(m))$.
4. Alice entschlüsselt und überprüft die Unterschrift: $V_B(D_A(E_A(S_B(m)))) = m$? Stimmen die Nachrichten überein, hat Bob die Nachricht korrekt bekommen.

Protokollattacke gegen Digitale Empfangsbestätigungen I

Wenn für die Verschlüsselung und die Unterschriftsprüfung der gleiche Algorithmus verwendet wird, gibt eine mögliche Attacke, denn dann gilt:

$$V_x = E_x \quad \text{und} \quad S_x = D_x$$

Mallory besitzt einen eigenen Private und Public Key. Er liest Bobs Mail und behält eine Nachricht von Alice an Bob (die er lesen will). Nach einiger Zeit schickt er diese an Bob mit der Behauptung, daß sie von ihm (also Mallory) käme. Bob entschlüsselt die Nachricht mit seinem Private Key und versucht Mallorys Unterschrift mit dessen Public Key zu überprüfen:

$$V_M(D_B(E_B(S_A(m)))) = E_M(S_A(m))$$

Protokollattacke gegen Digitale Empfangsbestätigungen II

Bob folgt dem Protokoll und schickt Mallory eine Empfangsbestätigung (receipt):

$$E_M(S_B(E_M(S_A(m))))$$

Mallory entschlüsselt nun mit seinem Private Key, verschlüsselt mit Bobs Public Key, entschlüsselt mit seinem eigenen Private Key und verschlüsselt mit Alices Public Key. Damit hat er die Nachricht m.
Das ganze Szenario ist ev. nicht unrealistisch, etwa beim Senden von automatischen Empfangsbestätigungen (receipt) - wo ist der Unterschied zur urspr. Idee ?
Abhilfe:

- Verwendung verschiedener Operationen (Algorithmus oder Key)
- Timestamps
- Unterschriften mit One-Way Hash Funktionen.

Protokollattacke gegen Digitale Empfangsbestätigungen III

HÜ2: Erklären sie detailliert die folgenden Fragen:

- An welcher Stelle des beschriebenen Angriffs ist tatsächlich die Bedingung
$$V_x = E_x \quad \text{und} \quad S_x = D_x$$
für den Erfolg notwendig ? Warum funktioniert das ohne diese Bedingung nicht ?
- Warum kann bei einer Unterschriftsleistung bei der vor der Unterschrift One-Way Hash Funktionen eingesetzt werden dieser Angriff vereitelt werden ? In welchem Schritt ?

Key Exchange mit symmetrischer Kryptographie

Häufig wird jede individuelle Kommunikation mit einem Session Key verschlüsselt. Die Verteilung dieses Schlüssels - Teil des Key Managements - kann sehr kompliziert sein. Alice und Bob teilen je einen geheimen Schlüssel mit dem Key Distribution Center (KDC, d.h. Trent).

1. Alice verlangt von Trent den Session Key, um mit Bob kommunizieren zu können.
2. Trent generiert einen zufälligen Session Key. Er verschlüsselt zwei Kopien: Einen mit Alices Key und einen mit Bobs Key. Trent schickt dann beides zu Alice.
3. Alice entschlüsselt ihre Kopie.
4. Alice schickt Bob seine Kopie.
5. Bob entschlüsselt seine Kopie.
6. Alice und Bob verwenden den Key, um verschlüsselt zu kommunizieren.

Key Exchange mit Public Key Kryptographie

Das vorige Verfahren hat einige bekannte Probleme:

- Die Sicherheit liegt ganz bei Trent.
- Trent ist ein Kommunikations “Bottleneck´´ und muss bezahlt werden.

Die Alternative mit Public Key Kryptographie ist attraktiver:

1. Alice holt Bobs Public Key vom KDC.
2. Alice generiert den Session Key, verschlüsselt ihn mit Bobs Public Key und schickt ihn an Bob.
3. Bob entschlüsselt Alices Nachricht mit seinem Private Key.
4. Beide verwenden den Session Key zur verschlüsselten Kommunikation.

Angriff auf Key-Exchange: Man in the Middle (MiM) Attack

1. Alice schickt Bob ihren Public Key (oder Bob holt diesen vom KDC). Mallory fängt den Key ab und schickt Bob ihren eigenen Public Key.
2. Bob schickt Alice seinen Public Key (oder Alice holt diesen vom KDC). Wie vorher fängt Mallory den Key ab und schickt Alice ihren eigenen Public Key.
3. Wenn Alice an Bob eine Nachricht schickt, fängt Mallory diese ab, entschlüsselt sie, liest sie, verschlüsselt mit Bobs Key und schickt das Resultat an Bob.
4. Wenn Bob an Alice etwas schickt, handelt Mallory analog.

Diese Attacke funktioniert auch, wenn die Schlüssel im KDC bereits manipuliert/ausgetauscht werden. Das Problem ist hier die **fehlende Schlüssel Authentifizierung**, d.h. es wird nicht überprüft ob die verwendeten Public Keys tatsächlich diejenigen der intendierten Zielpersonen sind.

Verhinderung der MiM Attacke

- **Generisch**: Schlüsselaustausch mit digitalen Signaturen / Zertifikaten (Trent bestätigt mit digitaler Unterschrift die Authentizität von Public Keys)
- **Speziell**: Interlock Protokoll
  1. Alice schickt Bob ihren Public Key.
  2. Bob schickt Alice seinen Public Key.
  3. Alice verschlüsselt ihre Nachricht mit Bobs Key und schickt die halbe Nachricht an Bob.
  4. Bob verschlüsselt seine Nachricht mit Alices Key und schickt die Hälfte an Alice.
  5. Alice schickt die zweite Hälfte.
  6. Bob setzt die beiden Hälften zusammen und entschlüsselt sie. Dann schickt er seine zweite Hälte.
  7. Alice setzt die beiden Hälften zusammen und entschlüsselt Bobs Nachricht.

Wie und Warum funktioniert das Interlock Protokoll ?

Die Idee / Annahme ist dass das verwendete Verschlüsselungsverfahren nur funktioniert, wenn ganze Nachrichten verschlüsselt/verschickt werden, aber halbe Nachrichten können nicht entschlüsselt werden. Dieses Verfahren vereitelt Mallorys Attacke. Wenn er in Punkt (3) die halbe Nachricht abfängt, kann er sie nicht entschlüsseln, da ja nur vollständige Nachrichten entschlüsselt werden können. Sie muß also eine neue Nachricht erfinden und weiterleiten (was vermutlich entdeckt werden würde).

Das Verfahren kann mit einem **Block Algorithmus** realisiert werden, bei dem jeder Nachrichtenblock hinsichtlich obiger Eigenschaft geteilt wird:

- Die Entschlüsselung hängt immer von einer Initialisierung ab, z.B. einem Initialisierungsvektor. Diese(r) wird erst in der zweiten Hälfte gesendet (könnte auch eine zweite Verschlüsselung sein deren Key in der zweiten Hälfte geschickt wird).
- Die beiden halben Blöcke enthalten folgenden Inhalt: Erste Hälfte: Even bits der verschlüsselten Nachricht, Zweite Hälfte: Odd bits.

Personen Authentifizierung

Wenn man sich an einem Computer anmeldet, muss man ein PWD angeben. Der Rechner muss natürlich die PWDs kennen um gültige von ungültigen unterschieden zu können. Aus Sicherheitsgründen (falls diese PWD Datei geleaked wird) werden diese nicht im Klartext sondern als Bild einer One-Way Funktion gespeichert:

1. Alice schickt ihr Passwort (remote oder am Terminal).
2. Computer wendet One-Way Funktion auf Passwort an.
3. Dann vergleicht er das Ergebnis mit seinen gespeicherten Daten.

Dadurch erscheint die Liste der Passwörter nutzlos, da sie nur Bilder unter der One-Way Funktion enthält. Aber auch in diesem Fall gibt es eine mögliche Attacke die – **Dictionary Attack**: Eine Liste von häufigen Passwörtern wird mit der (bekannten) One-Way Funktion verschlüsselt und mit den Einträgen in der Passwortliste verglichen. Diese Attacke ist sehr erfolgreich, da viele Menschen bei der Passwortgenerierung nicht sehr phantasievoll sind (Mausi1, Schatzi7), z.B. konnten 1989 mit einer Liste aus 732.000 PWDs 30% aller PWDs geknackt werden (Crypto 1989, Feldmeier/Karn).

Rainbow-Tables

Eine **Rainbow-Table** ist eine Datenstruktur die eine schnelle, speichereffiziente Suche nach einem PWD für einen Hashwert ermöglicht. Dabei werden PWD-sequenzen ("Ketten") gebildet, die mit einem PWD starten, das durch die one-way Funktion und schliesslich eine Reduktionsfunktion geleitet wird mit dem Ergebnis eines weiteren potentiellen PWD. Dieses Vorgehen wird iteriert, abgespeichert wird das erste PWD und das letzte PWD der Kette. Wichtig ist, dass kein PWD das in einer Kette vorkommt als Start-PWD verwendet wird, aber alle möglichen PWDs in der Tabelle vorkommen (wird durch periodisch wechselnde Reduktionsfunktionen erreicht).

Anwendung: Um ein PWD zu einem Hashwert zu finden, wird der Hash des gesuchten Kennworts durch die oben beschriebene Hash-Reduktion-Sequenz geführt, bis das Ergebnis der Reduktion in der Tabelle der letzten Kettenglieder vorkommt. Damit kennt man das entsprechende Start-PWD dieser Kette und kann von diesem ausgehend die Hash-Reduktion-Sequenz ausführen, um das gesuchte PWD zu erhalten.

- **Salt**: Das PWD wird nicht direkt in die one-way Funktion gespeist, sondern es wird vorher mit dem sog. Salt verbunden. Salt ist entweder für alle Nutzer gleich (was die Verwendung von Rainbow-Tables verunmöglicht weil die Abbildung der PWDs auf die Hashwerte anders ist), oder aber es wird für jeden Nutzer bei Anlegen des Accounts zufällig erstellt und beim Hashwert abgelegt. Im ersten Fall bleibt ein systematisches Durchtesten aller PWD noch möglich, im zweiten Fall ist das deutlich erschwert, weil zwei identische PWD unterschiedlicher Nutzer nicht den gleichen Hash liefert.
- **Pepper**: Pepper wird für alle User gleich oder unterschiedlich gewählt und **geheimgehalten** was Angriffe erschwert.

Ein “schwacher” Salt (z.B. Windows XP, Vista) stellt nur wenige verschiedene Werte zur Verfügung (z.B. 1000), sodass Rainbow-Tables noch sinnvoll sein können. Bei den erwähnten Systemen wird der UID als Salt verwendet, wodurch für “administrator” eine Rainbow-Table sehr nützlich ist.

SKEY Authentifizierung

Dieses System besteht aus einer One-Way Funktion f und einer zufälligen Zahl R.
$f(R), f(f(R)), \dots f^{100}(R) \Rightarrow x_1, x_2, \dots x_{100}$ werden berechnet.

Alice merkt sich die Zahlen und der Computer speichert x_{101}. Wenn Alice sich einloggen will, gibt sie x_{100} an. Der Computer berechnet $f(x_{100})$ und vergleicht das Ergebnis mit x_{101}. Wenn die beiden übereinstimmen, wird Alice akzeptiert. Der Computer ersetzt x_{100} und x_{101}. Alice streicht x_{100} von ihrer Liste.

Die meisten TAN Verfahren verlangen keine vorgegebene Reihenfolge, das Grundprinzip ist aber ähnlich.

Authentifizierung mit Public Key Kryptographie

Bietet eine Möglichkeit der Remote Authentifizierung an einem Rechner über ein unsicheres netzwerk. Als Vorraussetzung muss der Rechner Zugang zu den Public Keys der User haben, die Benutzer haben ihren Private Key.

1. Der Computer schickt Alice einen zufälligen String.
2. Alice verschlüsselt den String mit dem Private Key und schickt das Ergebnis mit ihrem Namen zurück (sie signiert also tatsächlich diesen String).
3. Der Computer entschlüsselt/verifiziert den String mit Alice's Public Key und vergleicht diesen mit dem Original. Bei Übereinstimmung ist Alice erfolgreich authentifiziert.

Problem: Wir werden sehen, dass es unsicher und gefährlich ist, irgendwelche zufälligen Strings zu Signieren / mit dem Private Key zu verschlüsseln.

Multiple Key Public Key Kryptographie I

Verallgemeinert das Konzept der Public Key Kryptographie: ist eine Teilmenge von Schlüsseln verwendet worden um eine Nachricht zu verschlüsseln, wird der Rest der Schlüssl benötigt um zu entschlüsseln.

Encryption	**Decryption**
K_A	$K_B \& K_C$
K_B	$K_A \& K_C$
K_C	$K_A \& K_B$
$K_A \& K_B$	K_C
$K_A \& K_C$	K_B
$K_B \& K_C$	K_A

Anwendung: Eine Gruppe von Leuten und man will mit vorher nicht bekannten Teilgruppen geheim kommunizieren. Dies bräuchte im klassischen Fall:

- Einen Schlüssel für jede mögliche Kombination $\rightarrow$ viele Schlüssel.
- Jede Nachricht muß für jede Kombination extra verschlüsselt werden $\rightarrow$ viele Nachrichten.

Alice	K_A
Bob	K_B
Carol	K_C
Dave	$K_A \& K_B$
Ellen	$K_B \& K_C$
Frank	$K_C \& K_A$

Das könnte etwa so aussehen:

Alice verschlüsselt mit K_A
$\Longrightarrow$ Ellen oder Bob & Carol gemeinsam können entschlüsseln.

Dave K_A wie Alice
K_B wie Bob
oder so daß nur Carol entschlüsseln kann.

Mit der Multiple Key Public Key Kryptographie ist die Handhabung einfacher.

Secret Splitting

Secret Splitting bedeutet, daß eine Nachricht in mehrere Teile aufgeteilt wird. Jede einzelne Teil ist für sich alleine bedeutungslos. Alle gemeinsam ergeben erst die Nachricht, wenn allerdings ein Teil verloren geht, ist die Nachricht verloren.

Anwendung: Aufteilen von Bauplänen, PWD für Tresore, Bilder splitten

Das Verfahren ist einfach. Trent splittet M zwischen Alice und Bob:

1. Trent generiert einen zufälligen String R gleich lang wie M.
2. Trent wendet ein XOR auf M und R an: $M \oplus R = S$
3. Trent schickt R an Alice und S an Bob.

Secret Splitting funktioniert auch mit mehreren TeilnehmerInnen:

1. Trent generiert drei Strings R, S, T.
2. $$M \oplus R \oplus S \oplus T = U$$

   usw.

Auch als (m, n)-threshold Verfahren bezeichnet: Die Nachricht wird in n Stücke (shadows) geteilt, sodaß mit m Teilen die Nachricht rekonstruiert werden kann. Wichtig dabei ist $m \leq n$!

Shamir's secret sharing realisiert das beruhend auf Modulararithmetik und wird nach der entsprechenden Wiederholung behandelt.

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- DES
- RSA
  - Mathematische Grundlagen: Zahlentheorie
  - RSA Algorithmus
- Kecchak / SHA-3

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit

Historisches zu DES

DES ist die Abkürzung für **Data Encryption Standard** und ist ein seit den 1970ern standardisiertes Blockcipher Verschlüsselungsverfahren. Der richtige Name ist eigentlich DEA, Data Encryption Algorithm nach ANSI in den USA oder DEA-1 nach ISO.

Anfang der 70er Jahre suchte das NBS (National Bureau of Standards) nach einem Verschlüsselungssystem, das standardisiert werden konnte, und der Öffentlichkeit zum Schutz wichtiger Daten zur Verfügung gestellt werden sollte.

1974 wurde das IBM Verfahren Lucifer für das NBS weiterentwickelt und später zum DES Standard erklärt. Der dabei verwendete Algorithmus war seit längerem der Öffentlichkeit bekannt, und mehrmals auf seine Zuverlässigkeit überprüft worden. Die Mitarbeit des NBS an seiner Weiterentwicklung ließ allerdings das Gerücht aufkommen, das NBS hätte heimlich eine Hintertür für den eigenen Gebrauch (Trapdoor) eingebaut.

Bis heute ungeklärt ob das so ist.

DES - von den 70ern bis Heute !

Am 23.11.1976 wurde der DES Standard schließlich veröffentlicht (!!).

Schon 1987 gab es Diskussionen um die Wiederzertifizierung, jedoch wurde DES sogar 1993 nochmals zertifiziert (obwohl es technologisch schon gar nicht mehr auf der Höhe der Zeit war, “never say not”).

1998 wiederholte sich die Ablösediskussion, Sachlage war zu klar, ein Verfahren zur Ablöse wurde initiiert (AES).

Aus heutiger Sicht ist DES durch die zu kleinen Parameter bei Blockgröße und Key ein **komplett unsicheres Verfahren**, das heute in keiner Form mehr verwendet werden soll/darf. NIST “verbietet” die Verwendung von TripleDES mit 3 Keys (3DES) ab 2023 for Federal Agencies, in neuen Applikationen seit 2017, trotzdem wird es aktuell im Bankenbereich zur B2B Kommunikation verwendet (Argument: untragbare Umstellungskosten weil in Hardware) und u.U. im Bereich CC und Smartcard-Terminal Kommunikation.

Wesentliche Eigenschaften von sicheren Ciphern (Shannon) I

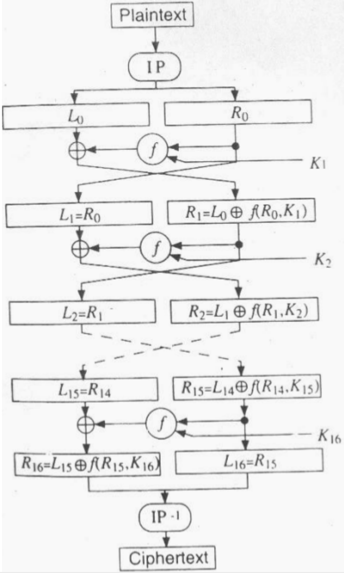
- **Confusion**: Confusion verschleiert die Beziehung zwischen Key und Ciphertext. Jedes Ciphertext Bit sollte von mehreren Schlüsselteilen abhängen, und wenn ein Schlüsselbit geändert wird, sollten die meisten oder alle Ciphertext Bits betroffen sein. Confusion wird durch Substitution erreicht und von Block- und Streamciphers verwendet.
- **Diffusion**: Eine Änderung im Plaintext (1 Bit) bewirkt eine Änderung an vielen Stellen (50%) des Ciphertextes. Das Ziel ist statistische Beziehungen zwischen Plaintext und Ciphertext zu verbergen, z.B. sollen redundante Muster in Plaintext im Ciphertext nicht erkennbar sein. Diffusion wird durch Permutation erreicht und nur in Blockciphers verwendet. *Strict avalanche criterium (SAC)* nach Feistel (s.u.) besagt dass wenn 1 Plaintext Bit geändert wird, ändert sich jedes Ciphertext Bit mit 50% Wahrscheinlichkeit (in der Chaos Theorie auch als "Butterfly Effekt" bekannt).

Wesentliche Eigenschaften von sicheren Ciphern (Shannon) II

Häufig werden Confusion und Diffusion Komponenten kombiniert um erhöhte Sicherheit gegen Kryptoanalyse vergleichen mit den Einzelkomponenten zu erzielen - *Product Cipher*.

Ein klassisches Beispiel sind *Feistel Ciphers (oder Networks)*, bei denen Ver- und Entschlüsselung sehr ähnlich ablaufen und beide Operationen aus einer iterative Ausführung einer "Rundenfunktion" (round) bestehen.

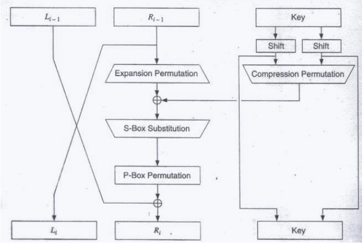
Die meisten jemals eingesetzten Blockcipher folgen diesem Designprinzip, neben DES auch e.g. AES, GOST, IDEA, u.s.w.



DES ist ein symmetrisches Blockcipher Verfahren mit 64 Bit Blockgrösse und 56 Bit effektiver Schlüssellänge. Die 64 Schlüsselbits enthalten 8 Bits zur Fehlererkennung.

Nach einer initialen Permutation wird der Block in zwei Hälften geteilt (je 32 Bit). In den 16 Runden identischer Operationen (Feistel Cipher/Network) werden die Daten mit dem Schlüssel kombiniert. Am Ende werden die beiden Hälften wieder zusammengeführt und eine finale Permutation beendet den Algorithmus.

Das Kernstück des DES Verfahrens - der “fundamental building block´´ kombiniert Confusion und Diffusion unter Verwendung eines Schlüssels in einer Schleife (ein Product Cipher).



In jeder Runde wird der Schlüssel geshiftet und von 56 Bit 48 ausgewählt. Die rechte Hälfte der Daten wird auf 48 Bits vergrößert (Expansion Permutation) und ein XOR auf diese Daten und den geshifteten und permutierten Schlüssel angewendet. Die resultierenden Daten werden durch 8 sogenannte S-Boxen geschickt (= 32 Bits) und dann nochmals permutiert. Dieser gesamte Vorgang wird mit der Funktion f dargestellt. Auf das Ergebnis von f und die linke Hälfte der Daten wird dann ein XOR angewendet. Das Ergebnis dieser Operation wird dann die neue rechte Hälfte. Die alte rechte Hälfte wird die neue linke Hälfte.

DES Komponenten: Schlüsselerzeugung

Die initiale als auch die finale Permutation betreffen nicht die Sicherheit sondern werden durchgeführt, um Plain- und Ciphertext Daten effizient auf DES Chips laden zu können.

Die 56 Schlüsselbits werden zweigeteilt und die Hälften jeweils zirkulär in Abhängigkeit von der aktuellen Schleifennummer nach links geshiftet.

Number of Key Bits Shifted per Round

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

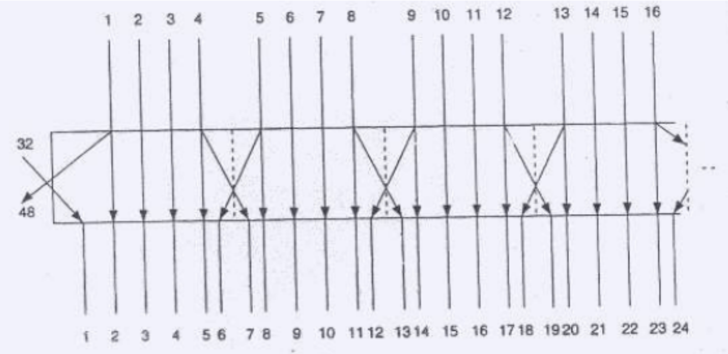
Anschließend weden 48 Bits ausgewählt und permutiert (Compressions Permutation). Das Bit 18, etwa, wird ignoriert. Auf Grund des Shiftens wird in jedem Subkey eine andere Teilmenge der Key Bits verwendet.

Compression Permutation

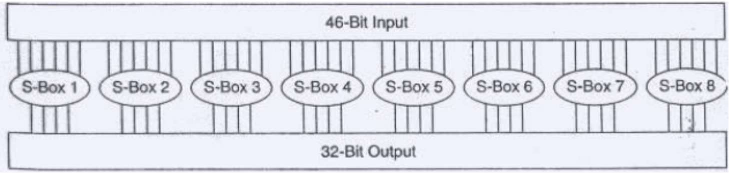
14,	17,	11,	24,	1,	5,	3,	28,	15,	6,	21,	10,
23,	19,	12,	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32

Die rechte Hälfte der Daten wird von 32 auf 48 Bits erweitert. Dies hat zwei Gründe:

- Die Daten sind gleich lang wie der Key was ein XOR ermöglicht und können dann bei der Substitution wieder komprimiert werden.
- Manche Bits betreffen dann zwei Substitutionen, d.h. die Abhängigkeit der Output Bits von den Input Bits weitet sich schneller aus. Dies erzeugt den Lawineneffekt.



Nach der XOR Operation mit dem Schlüssel wird das 48 Bits lange Ergebnis einer Substitution unterworfen.



Jede S-Box hat 6-Bit Input und 4-Bit Output. Es gibt acht verschiedene S-Boxen. Die 48 Bits werden in 6-Bit Blöcke aufgeteilt. Jede S-Box hat 4 Zeilen (0-3) und 16 Spalten (0-15). Jeder Eintrag ist eine 4-Bit Zahl. Die 6 Input Bits bestimmen, wo in der Tabelle nachgeschaut werden muß.

$b_1, \ldots, b_6$:	b_1, b_6	laufen von 0-3:	zeile
	b_2, b_3, b_4, b_5	laufen von 0-15:	spalte

DES-Komponenten: S-Box Beispiel

Im folgenden Beispiel ist die 6. S-Box dargestellt:

S-box 6:

12,	1,	10,	15,	9,	2,	6,	8,	0,	13,	3,	4,	14,	7,	5,	11,
10,	15,	4,	2,	7,	12,	9,	5,	6,	1,	13,	14,	0,	11,	3,	8,
9,	14,	15,	5,	2,	8,	12,	3,	7,	0,	4,	10,	1,	13,	11,	6,
4,	3,	2,	12,	9,	5,	15,	10,	11,	14,	1,	7,	6,	0,	8,	13,

$$110011: \quad 11 \rightarrow 3$$

$$1001 \rightarrow 9 \quad \rightarrow 14$$

$$\downarrow$$

$$1110$$

Die S-Box Substitutionen sind keine linearen Operationen und für die Sicherheit von DES von zentraler Bedeutung. (Falls es wirklich eine Hintertür für dieses Verfahren geben sollte, wäre dies der geeignete Platz – in der Tat wurden bei den S-Boxen durch das NBS Änderungen gegenüber dem original Lucifer vorgenommen.)

DES - Kompenenten: finale Permutation und Entschlüsselung

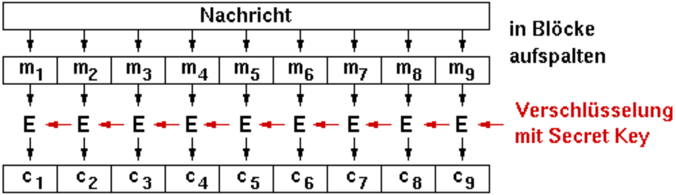
Nach der Substitution (S-Box) wird eine Permutation auf die 32-Bit Daten angewendet und auf das Ergebnis mit der linken Hälfte des ursprünglichen 64-Bit Blockes wird ein XOR angewendet.

Die finale Permutation ist invers zur initialen Permutation, somit kann der Algorithmus sowohl zum Ent- als auch zum Verschlüsseln verwendet werden.

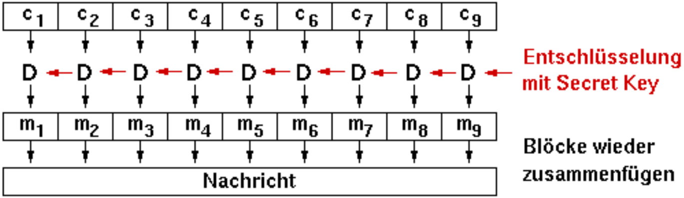
Die Entschlüsselung funktioniert analog zur Verschlüsselung, nur daß die Schlüssel in der umgekehrten Reihenfolge verwendet werden. Auch der Key-Generierungsalgorithmus ist zirkulär, nur daß hier ein Rechts Shift verwendet wird (# Shifts per Position: 0,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1).

DES ist insbesondere auf Geschwindigkeit in Hardware optimiert - schon in den 90ern gab es einen experimentellen DEC/Compac Chip der 16.8 Mio Blöcke pro Sekunde (1 Gigabit/Sekunde) verarbeiten konnte.

Nach der Aufteilung der Nachricht in Blöcke (blocksize cipher) wird ein Block Plaintext in einen Block Ciphertext überführt.



Die Entschlüsselung folgt trivialerweise dem gleichen Vorgang.



Blockcipher Betriebsmodi: ECB Eigenschaften

- Vorteile
  1. Die Reihenfolge der Blöcke muss bei der Ver- bzw. Entschlüsselung nicht eingehalten werden - optimal für Datenbanken oder Parallelisierung.
  2. Fehler im Plaintext betreffen nur einen spezifischen Ciphertext Block. Wenn Blockgrenzen kontrolliert werden, ist auch ein Fehler von Ciphertext Bits kein Problem - keine Ausbreitung.
- Nachteile
  1. Es kann ein Codebook aller Plaintext- Ciphertext Paare erstellt werden, v.a. problematisch bei Stereotypen am Anfang und am Ende von Nachrichten (Begrüssung, Fileheader)
  2. Block Replay: siehe nächste Folie ! **Abhilfe:** Die Schlüssel sollten oft gewechselt werden, denn dann muß ein Angreifer schneller sein.

Blockcipher Betriebsmodi: Block Replay bei ECB

Mallory kann die verschlüsselten Nachrichten verändern, ohne daß es bemerkt wird.

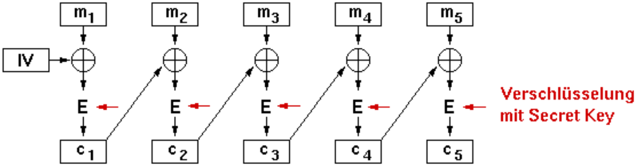
z.B.:	B_1	Senderbank	$1,5$ Blocks
	B_2	Empfängerbank	$1,5$ Blocks
	Einzahlender		6 Blocks
	Konto		2 Blocks
	Summe		1 Blocks

Ein Block besteht dabei aus z.B. 8 Byte. Mallory zeichnet die B2B Kommunikation zwischen Banken auf und transferiert selbst 100 $ von B1 nach B2 und wiederholt das später. Er analysiert seine Aufzeichnungen auf zwei identische Nachrichten, die er findet. Nun kann er diese Nachricht nach Belieben einspielen und bekommt jedes Mal 100 $.

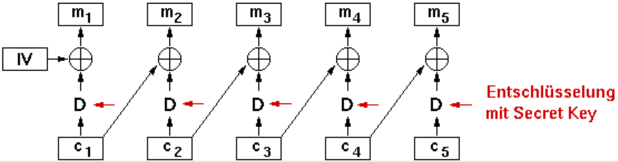
Abhilfe: Time stamp davor: 1 Block zusätzlich – Mallory kann allerdings herausfinden, wo es in den Nachrichten um seinen Namen geht und in jede Nachricht Originaldaten mit seinen verschlüsselten Daten - Name, Summe, etc. - ersetzen.

Blockcipher Betriebsmodi: Cipher Block Chaining (CBC)

Ein zusätzlicher Feedback Mechansimus wird eingefügt: auf den Plaintext Block wird mit dem vorhergehenden Ciphertext Block ein XOR angewendet, ehe er verschlüsselt wird: $C_i = E_K(P_i \oplus C_{i-1})$. Ein random Initialisierungsvektor (IV) ist der erste Block – identische Plaintexte werden zu verschiedenen Ciphertexten.



Entschlüsselung: $P_i = C_{i-1} \oplus D_K(C_i)$



Blockcipher Betriebsmodi: CBC Eigenschaften

Ohne IV werden zwei identische Texte werden gleich verschlüsselt. Wenn sie sich ab einem bestimmten Punkt unterscheiden, unterscheiden sich auch die beiden Ciphertexte an der genau gleichen Stelle.

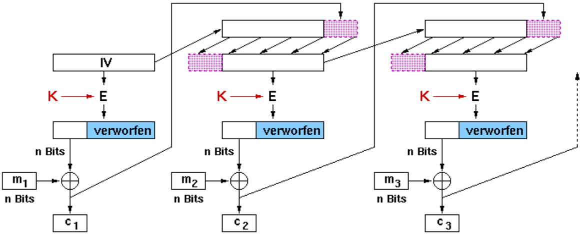
Fehlerausbreitung:

- *Plaintextfehler*: Ein Fehler im Plaintext betrifft zwar den gesamten Ciphertext, beim Entschlüsseln bleibt aber trotzdem nur der ursprüngliche Fehler zurück.
- *Ciphertextfehler*: Ein fehlerhaftes Bit (Channel oder Storage Error) zerstört den gesamten Plaintext Block und ein Bit des nächsten Plaintext Blocks an der Fehlerstelle (error extension). Der zweite Block nach dem fehlerhaften Bit ist allerdings wieder korrekt (“self recovering´´).

HÜ3: Erklären sie detailliert wie es zu den beschriebenen Fehlern nach einem Ciphertext Bitfehler kommt und warum CBC self recovering ist.

Blockcipher Betriebsmodi: Cipher Feedback Mode (CFB)

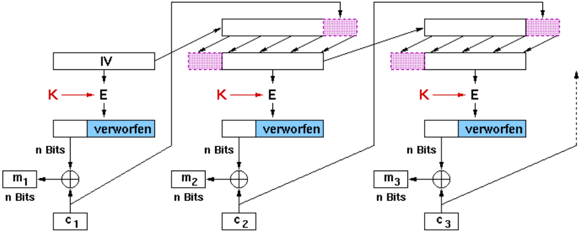
Block Ciphers werden auch als “self-synchronizing stream ciphers´´ verwendet. Hier genügt es, wenn *n* Bit Plaintext verfügbar sind.



Eine Queue die genau Blockgröße hat wird wiederholt verschlüsselt, zu Beginn wird sie mit einem IV gefüllt, verschlüsselt und auf die *n*-Bit am linken Rand der Queue wird mit dem Plaintext ein XOR angewendet. Dieser Ciphertext wird rechts in die Queue geschoben, was *n* Bits aus der Queue schiebt. Dann kommt der nächste *n*-Bit Block an die Reihe.

Blockcipher Betriebsmodi: CFB Eigenschaften

Die Entschlüsselung ist genau umgekehrt (Rollen von Plain- und Ciphertext vertauscht), aber ebenfalls im Verschlüsselungsmodus (das gleiche “OTP” muss erzeugt werden).

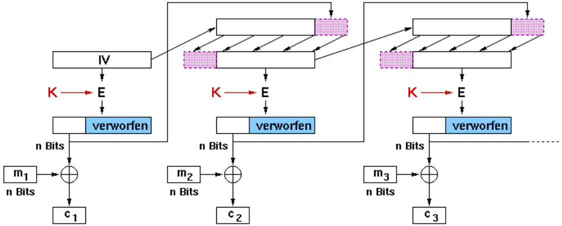


Fehlerausbreitung:

- *Plaintextfehler*: Analog zum CBC Mode betrifft ein Plaintextfehler den gesamten Ciphertext ab dieser Stelle. Nach der Entschlüsselung wieder auf die Ausgangsstelle reduziert.
- *Ciphertextfehler*: Ein Ciphertext Bitfehler verursacht Fehler im Plaintext solange es sich im Register befindet. Nach dem “Hinausfallen´´ des Fehlers erholt sich das System wieder.

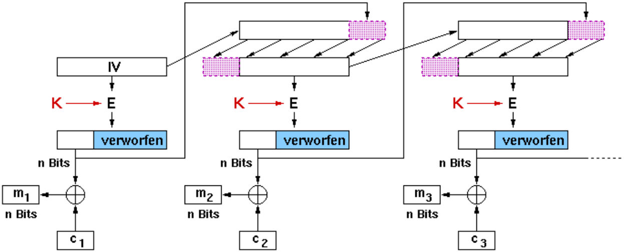
Blockcipher Betriebsmodi: Output Feedback Mode (OFB)

Das ist eine Methode um einen Block Cipher als synchronen Stream Cipher zu verwenden. Das Verfahren ist dem CFB sehr ähnlich und verwendet ebenfalls eine Queue mit Feedback. Allerdings werden hier *n*-Bits der verschlüsselten Queue wieder in die Queue geschoben (und nicht der Ciphertext) vor der nächsten Verschlüsselung desselben.



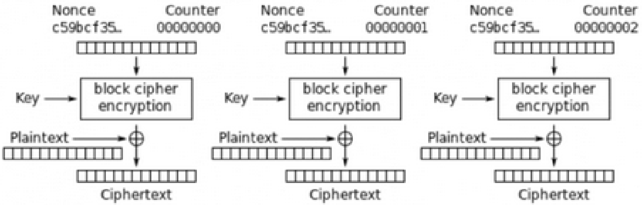
Der gesamte Verschlüsselungsvorgang kann geschehen, bevor der Plaintext vorhanden ist, da nur noch ein XOR angewendet wird. Da das Feedback u.a. von Plain- und Ciphertext ist wird OFB auch als "internal feedback" bezeichnet.

Die Entschlüsselung erfolgt wieder genau umgekehrt zur Verschlüsselung, es muss nur das gleiche "OTP" erzeugt werden.



- Hier gibt es keine Fehlerausbreitung. Ein Bit-Fehler im Plain- bzw. Ciphertext bewirkt einen Bit-Fehler im Cipher- bzw. Plaintext.
- *Sicherheit*: OFB sollte nur verwendet werden, wenn die Feedback Größe genau der Block Größe entspricht (full block feedback). Wenn die Größen übereinstimmen, permutiert der Block Cipher die Queue Werte und die Periodenlänge ist im Schnitt $2^m - 1$, mit m als Blockgröße. Sonst wird die Periodenlänge kleiner.

Hier wird ebenso ein Keystream produziert der mit dem Plaintext geXORed wird. Dies geschieht durch Verschlüsselung einer Zählers, der meist mit einem IV ("Nonce") kombiniert ist. Dadurch sind Teile des Keystreams unabhängig voneinander und Verschlüsselung kann parallel durchgeführt werden und erlaubt auch random access. Wie beim OFB kann der Keystream vorberechnet werden.



- Keinerlei Fehlerausbreitung (wie bei OFB).
- *Sicherheit*: kein wiederholter Einsatz von identischen IV und Schlüsseln (OTP Eigenschaften !!).

Sicherheit von DES - Keys

Weak Keys: Durch das Halbieren und u.a. Shiften bleiben die Schlüssel gleich, die nur aus 0, 1 oder zur Hälfte aus 0, 1 bestehen.

Semiweak Key Pairs: Schlüsselpaare, die das gleiche Ergebnis liefern.

Possible Weak Keys: geben in den Keyshifts nur vier Schlüssel.

Complement Keys: Das Bitweise Komplement eines Schlüssels verschlüsselt das Komplement des Plaintext Blocks ist das Komplement des Ciphertextblocks. Sei X das Original und X' das Komplement:

$$E_K(P) = C$$

$$E_{K'}(P') = C'$$

Bei einer chosen Plaintext Attacke muss daher nur halbe Schlüsselanzahl getestet werden (2^{55} statt 2^{56}). Dieser Effekt entsteht durch das XOR der Subkeys nach der Expansionspermutation.

Schlüssellänge: Mit dem Vorhandensein von leistungsfähigeren Computern werden immer längere Schlüssel benötigt. 1993 wurde gezeigt, daß mit einem Computer im Wert von etwa einer Million Dollar eine Brute Force Attacke in $3,5$ Stunden ausführbar ist, solche Rechner wurden definitiv gebaut. **Achtung**: Moore's Law !!

S-Box Design: Detaillierte Analysen zeigen, dass besonders starke und auch auffällig schwache Strukturen vorhanden sind. Die NBS Änderungen könnten auch ev. eine IBM Hintertür entfernt haben. S-Boxen wurden gegen eine Differential Cryptoanalysis Attacke optimiert – dies hilft allerdings nicht gegen eine lineare Attacke. Es gibt bessere "Kompromiss-Boxes" (s^4DES & s^5DES), ausserdem verbessern Key-abhängige (nicht zufällige !) S-Boxes die Sicherheit (ursprüngliche S-Boxes werden umsortiert und innerhalb der Boxen werden schlüsselabhängig Zeilen und Spalten getauscht).

Blockälften: Daten- oder Schlüsselabhängiges Tauschen der Blockhälften (statt der fixen Vorgehensweise) verbessert die Sicherheit (RDES-1, . . . , RDES-4).

Anzahl der Runden: 1982 wurde DES mit 3 und 4 Runden gebrochen, 1988 DES mit 6 Runden. Jeder DES mit weniger als 16 Runden kann mit “Differential Cryptoanalysis´´ weitaus effizienter als mit einer Brute Force Attacke gebrochen werden. Das legt natürlich nahe, dass (1) die Entwickler von DES die Technik der Differential Cryptoanalysis kannten und (2) sie auch obiges Ergebnis bzgl. DES kannten (die Definition mit 16 Runden ist ziemlich sicher kein Zufall).

Ist eine Chosen Plaintext Attacke, die 1990 (wieder)entwickelt wurde. Es werden Paare von Ciphertexten benutzt, deren Plaintexte spezifische Unterschiede haben (spezifische Ausgangsdifferenzen, die nicht mit erwarteter Häufigkeit auftreten) - es werden Differenzen zwischen Eingabe- und Ausgabeblöcken betrachtet und diese über die Runden hinweg verfolgt (die betrachteten Differenzkominationen schränken den für einen Angriff relevanten Keyspace signifikant ein).

DES mit 16 Runden ist relativ resistent gegen diese Technik, da die Designer von DES vermutlich diese Attacke kannten. Mit 19 oder mehr Runden, wird die Attacke theoretisch unmöglich, weil mehr als 2^{64} Plaintexte benötigt würden.

Abhilfe: Die wichtigste Massnahme gegen diesen Angriff ist die Gestaltung der S-Boxen, sodass Ausgangsdifferenzen möglichst gleichverteilt auftreten.

Linear Cryptoanalysis

Ist eine Known Plaintext Attacke, die 1992 entwickelt wurde. Hauptstrategie ist es, in den nicht-linearen Teilen eines Ciphers (meist die Substitution) lineare Bestandteile ausfindig zu machen und daraus einige Schlüsselbits zu ermitteln:

$$\left\{\begin{array}{l}\text{XOR von Plaintext Bits}\\ \text{XOR von Ciphertext Bits}\end{array}\right\}\text{XOR}$$

Das Ergebnis ist ein XOR von einigen Key-Bits. Anschliessend kann geprüft werden, welche der gefundenen Gleichungen besonders viele Bit-Belegungen hat, bei denen die Gleichung stimmt. Aus diesen Gleichngen können involvierte Schlüsselbits mit einer bestimmten Wahrscheinlichkeit (wieviele Bit-Belegungen sind korrekt in %) berechnet werden. Eventuell werden mehrere Gleichungen pro Schlüsselbit verwendet, da die Wahrscheinlichkeit oft nahe bei 1/2 liegt. Dies bedeutet dass für jede Runde möglichst gute lineare Approximationen gefunden werden müssen.

Abhilfe: Gestaltung der S-Boxen, sodass diese möglichst wenig lineare Bestandteile aufweisen.

Sicherheit von DES - Algebraische Struktur

64-Bit Plaintext Blocks ergeben 64 Bit Ciphertext Blöcke auf 2^{64} verschiedene Arten. DES benutzt 2^{56} (wegen 56 Bit key, etwa 10^{17}) davon.

Idee: Mit mehrfacher Verschlüsselung scheint es möglich zu sein, einen größeren Anteil auszunutzen.

Ist DES eine abgeschlossene Gruppe (im algebraischen Sinn bzgl. Hintereinanderausführung der Verschlüsselung mit verschiedenen Keys), gibt es für jedes Schlüsselpaar K_1 und K_2 ein K_3, sodaß

$$E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$$

Dies würde bedeuten, dass eine Mehrfachverschlüsselung nicht profitabel ist, weil sie durch Einfachverschlüsselung mit einem anderen Key dargestellt werden kann.

1992 wurde allerdings bewiesen, daß DES keine solche Gruppe darstellt.

$$C = E_{K_2}(E_{K_1}(P))$$
$$P = D_{K_1}(D_{K_2}(C))$$

"**Meet in the middle Attacke**´´: CryptoanalystIn kennt P_1, C_1, P_2, C_2, sodaß

$$C_1 = E_{K_2}(E_{K_1}(P_1))$$
$$C_2 = E_{K_2}(E_{K_1}(P_2))$$

Für alle möglichen K (K_1 oder K_2) wird $E_K(P_1)$ berechnet und im Speicher abgelegt. Danach wird $D_K(C_1)$ für alle K berechnet und das identische Ergebnis im Speicher gesucht. Wird es gefunden, kann der aktuelle Schlüssel K_2 sein und der im Speicher K_1. Ergibt sich $C_2 = E_{K_2}(E_{K_1}(P_2))$, sind die Schlüssel zeimlich sicher gefunden.

Speicherbedarf: 2^n Blöcke mit n Keylength - 2^{56} 64-Bit Blöcke bei DES, d.h. 10^{17} Bytes.
Komplexitätsreduktion: 2^{112} Schlüsselpaare (exhaustive search) $-> 2^{57} = 2 * 2^{56}$ full search single Key für Ver- und Entschlüsselung.

Triple DES (mit 2 Keys)

$$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_1}(C)))$$

Dieses Verfahren wird auch EDE (encrypt-decrypt-encrypt mode) genannt. Während der gewöhnliche DES Block Algorithmus einen n-Bit Key hat, benutzt dieser einen $2n$-Bit Key. K_1 und K_2 wechseln sich ab, um eine Meet in the Middle Attacke auszuschließen. EDE wird verwendet um Rückwärtskompatibilität zu DES zu erhalten, nimmt man $K_1 = K_2$ so erhält man single DES mit K_1.

Frage: ist dieses Verfahren nun so sicher wie man vermuten würde ?

Triple DES (mit 2 Keys): Angriff

Es gibt hier auch eine Known Plaintext Attacke (Oorschot, Wiener, Crypto 1991):

1. Gegeben eine Tabelle mit Plaintext- und Ciphertext Paaren (Tabelle 1, sortiert nach Plaintexten).
2. Erraten/wählen Sie einen fixen ersten Zwischenwert $a = E_{K_1}(P)$.
3. Bestimmen sie durch die Berechnung von $P = D_{K_1}(a)$ den Plaintext und suchen den dazugehörenden Ciphertext C in Tabelle 1, dies für alle K_1.
4. Tabulieren Sie für alle K_1 (für das fixe a) den zweiten Zwischenwert, $b = D_{K_1}(C)$ (Tabelle 2).
5. Suchen Sie in der Tabelle 2 für alle möglichen K_2 Elemente mit passendem zweiten Zwischenwert b:
   $b = D_{K_2}(a)$.
6. Bei Übereinstimmung ist ein mögliches Schlüsselpaar gefunden (mit weiteren Plaintext / Ciphertextpaaren verifizieren). Ansonsten wird ein neues a gewählt.

Das funktioniert weil (angwendet auf Definition der Verschlüsselung s.o.):

$$D_{K_1}(C) = D_{K_1}(E_{K_1}(D_{K_2}(E_{K_1}(P))))$$

$$D_{K_1}(C) = D_{K_2}(E_{K_1}(P))$$

Die Erfolgswahrscheinlichkeit für ein a ist $\frac{p}{m}$, wobei p die Anzahl der bekannten Plaintext/Ciphertextpaare, m die Blockgröße und n der Keyspace ist.
Die insgesamt erwartete Laufzeit ist damit $\frac{2^{n+m}}{p}$, d.h. bei DES mit $p > 256$ ist $\frac{2^{120}}{p}$ schneller als eine vollständige Suche mit 2^{112}.

HÜ4: Erklären sie die Angriffskomplexität für DES, insbesondere betrachtend die Anzahl der im Mittel notwendigen a Versuche (die Originalarbeit gibt hier wesentliche Hinweise).

Triple DES (mit 3 Keys): 3DES

Ist nun (endlich) so sicher wie man glaubt daß die Double Encryption ist.

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_3}(C)))$$

denn:

$$D_{K_3}(C) = D_{K_3}(E_{K_3}(D_{K_2}(E_{K_1}(P))))$$
$$D_{K_3}(C) = D_{K_2}(E_{K_1}(P))$$

Für die meet in the middle attack berechne ich nun links 2^{56} Entschüsselungen und rechts 2^{112} Ver- und Entschlüsselungen um die Vergleiche zum Auffinden der zusammenpassenden Keys durchführen zu können. Das ist gleich aufwändig wie die exhaustive search.

3DES Encryption Modes

Inner CBC: Die gesamte Nachricht wird im CBC Mode dreimal verschlüsselt. Es werden drei verschiedene Initialisierungsvektoren (C_0, S_0, T_0) benötigt.

$$C_i = E_{K_3}(S_i \oplus C_{i-1})$$
$$S_i = D_{K_2}(T_i \oplus S_{i-1})$$
$$T_i = E_{K_1}(P_i \oplus T_{i-1})$$

Outer CBC: Die gesamte Nachricht wird im CBC Mode dreimal verschlüsselt – das ist die sicherste Variante.

$$C_i = E_{K_3}(D_{K_2}(E_{K_1}(P_i \oplus C_{i-1})))$$
$$P_i = C_{i-1} \oplus D_{K_1}(E_{K_2}(D_{K_3}(C_i)))$$

ECB-CBC Kombination: Weiters gibt es auch Kombinationen zwischen ECB, CBC in verschiedenen Variationen.

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- DES
- RSA
  - Mathematische Grundlagen: Zahlentheorie
  - RSA Algorithmus
- Kecchak / SHA-3

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit

Public-Key Verfahren beruhen nicht auf geheimen Schlüsseln sondern auf sogenannten “computational unfeasible problems´´. Das sind Probleme, die in der Komplexitätstheorie z.B. als “NP-complete´´ eingestuft werden.

$a \equiv b \pmod{n}$	“a kongruent b modulo n´´ - Das ist die Schreibweise für $a = b + kn$, $k \in Z$. Ist a nicht negativ und $0 \leq b < n$ kann b als Divisionsrest von $\frac{a}{n}$ aufgefaßt werden. n wird als Modul, b als Residuum bezeichnet.
$0 \dots n-1$	ist ein komplettes Restklassensystem modulo n (alle Residuen mod n, set of residues).
$a \bmod n$	Operation, die das Residuum von a bezüglich n liefert.
$17 \equiv 2 \pmod{3}$	$17 = 2 + 5 \cdot 3$, $17 \bmod 3 = 2$.

Modulararithmentik: Basics 2

Das Ergebnis der modularen Reduktion ist positiv. In den meisten Programmiersprachen sind allerdings auch negative Ergebnisse zugelassen. (Daher immer prüfen!).

Modulararithmetik ist kommutativ, assoziativ und distributiv. Jedes Zwischenergebnis kann in jedem beliebigen Schritt modular reduziert werden, das Ergebnis bleibt gleich.

Es gibt diverse Gründe, warum die Modular Arithmetik für die Kryptographie sehr interessant ist:

- Diskrete Logarithmen und Quadratwurzelberechnungen $\bmod n$ sind “computationally infeasible´´.

HÜ5: Erklären sie warum die Berechnung von Diskrete Logarithmen ($a^x \bmod n$ ist leicht, die Umkehrung – Suche x, sodaß $a^x \equiv b \pmod{n}$) ist sehr viel schwieriger) und Quadratwurzeln in Modulararithmetik “computationally infeasible´´ ist, während in klassischer Arithmetik die entsprechenden Berechnungen nicht besonders aufwändig sind.

Für ein k-Bit Modul n ist das Zwischenergebnis jeder Addition, Subtraktion und Multiplikation auf $2k$ Bits beschränkt.
$\rightarrow$ Exponentation kann in der Modular Arithmetik ohne große Zwischenergebnisse durchgeführt werden.

$a^8 \bmod n : (a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a \cdot a) \bmod n$
besser: $((a^2 \bmod n)^2 \bmod n)^2 \bmod n$
$a^x \bmod n$, wenn $x \neq 2^n$: siehe folgendes *Beispiel*:

$$\begin{aligned}
x &= 25 = 2^4 + 2^3 + 2^0 \\
a^{25} \bmod n &= (a \cdot a^{24}) \bmod n \\
&= (a \cdot a^8 \cdot a^{16}) \bmod n \\
&= (a \cdot ((a)^2)^2)^2 \cdot (((a^2)^2)^2)^2) \bmod n \\
&= ((((a^2 \cdot a)^2)^2)^2 \cdot a) \bmod n
\end{aligned}$$

Mit gespeicherten Zwischenergebnissen benötigt die Berechnung nur sechs Multiplikationen.

Mudulararithmetik: Primzahlen

Eine Primzahl ist eine positive ganze Zahl größer als 1, deren einzige Faktoren 1 und die Zahl selbst sind. Keine andere Zahl teilt eine Primzahl ohne Rest. Es gibt unendlich viele Primzahlen – $\sharp Primzahlen \leq n \sim \frac{n}{\log n}$, z.B. gibt es ungefähr$10^{151}$ Primzahlen mit 512 oder weniger Bits. Public-key Kryptographie benutzt oft grosse Primzahlen.

Wie findet man grosse Primzahlen ? Es ist sehr aufwändig, eine Zahl zu nehmen, eine Faktorisierung zu versuchen und bei Fehlschlag annzunehmen, daß es eine Primzahl ist.
$\rightarrow$ Es gibt probalistische Primzahl Tests, die mit hoher Wahrscheinlichkeit bestimmen, ob es sich um eine Primzahl handelt oder nicht.

Größter Gemeinsamer Teiler (GGT) – zwei Zahlen heißen relativ prim, wenn sie außer 1 keine gemeinsamen Faktoren haben ("teilerfremd´´, $(a, b) = 1$). In anderen Worten: Der größte gemeinsame Teiler von *a* und *b* ist 1.

z.B. $(3, 5) = 1$, $(3, 6) = 3$
Eine Primzahl ist zu allen anderen Zahlen außer ihren eigenen Vielfachen relativ prim.

Euklidischer Algorithmus ist das klassische Verfahren zur Berechnung des GGT von *x* und *y* (Voraussetzung: $x, y > 0$):

```
    while (x > y)
    {
        g = x;
        x = y % x;
        y = g;
    }
   return g;
```

Inversion in Modulararithmetik

In der klassischen Arithmetik ist 4 invers $\frac{1}{4}$, weil $4 \cdot \frac{1}{4} = 1$.

In der modularen Arithmetik $4 \cdot x \equiv 1 \pmod{n} \Leftrightarrow$ gesucht sind x und $k \in Z$, sodaß $4x = kn + 1$.

Wann existiert eine eindeutige Lösung für $a^{-1} \equiv x \pmod{n}$? Wenn $(a, n) = 1$.

HÜ6: Erklären sie (und geben sie ein Beispiel), wie modulare Inversion mit dem erweiterten Euklidischen Algorithmus realisiert werden kann.

Kleiner Satz von Fermat ("Kleiner Fermat´´): Sei p eine Primzahl und a kein Vielfaches von p.
$\Rightarrow a^{p-1} \equiv 1 \pmod{p}$

Beispiel: $p = 3$, $a = 2$
$$\begin{array}{lllll} 2^2 & = & 4 & \equiv & 1 \pmod{3} \\ 7^2 & = & 49 & \equiv & 1 \pmod{3} \end{array}$$

Eulersche Phi-Funktion: Das reduzierte Restklassensystem modulo n ist jene Teilmenge des kompletten Restklassensystems deren Residuen relativ prim zu n sind. Ist n eine Primzahl, so ist reduziert = komplett, nur ohne 0.
Die Eulersche Phi-Funktion $\phi(n)$ gibt die Anzahl der Elemente des reduzierten Resklassensystems modulo n an. Dies ist gleichbedeutend mit $\phi(n) = \#m \in N$, $m < n$ mit $(m, n) = 1$.

Ist n eine Primzahl: $\phi(n) = n - 1$.
Ist $n = p \cdot g$ mit p und g Primzahlen: $\phi(n) = (p - 1)(g - 1)$.

$(a, n) = 1 \Rightarrow a^{\phi(n)} \equiv 1 \bmod n$ bzw. $a^{\phi(n)} \bmod n = 1$.

Wir hatten für Inverses: $(a \cdot x) \bmod n = 1$
$\Rightarrow x = a^{\phi(n)-1} \bmod n$ ist Inverses zu a.

Als Beispiel rechnen wir das Inverse von $5 \pmod 7$:
$\phi(7) = 7 - 1 = 6$, $5^{6-1} \bmod 7 = 3$
Das Inverse ist also 3.

Was ist eine Primitivwurzel ? Ist p eine Primzahl und $q < p$, so ist q eine Primitivwurzel (primitives Element) PW modulo p, wenn

$$\forall b \;\; 1 \leq b \leq p-1 \;\; \exists a, \;\; \text{sodaß} \;\; q^a \equiv b \pmod p.$$

Es können mit q also alle Zahlen bis p durch potenzieren erzeugt werden.

Sei p eine Primzahl. a ist ein quadratisches Residuum von p, wenn $x^2 \equiv a \pmod{p}$ für beliebige x. Beispiel: $p = 7$

$$\begin{array}{lclcl} 1^2 & = & 1 & \equiv & 1 \pmod{7} \\ 2^2 & = & 4 & \equiv & 4 \pmod{7} \\ 3^2 & = & 9 & \equiv & 2 \pmod{7} \\ 4^2 & = & 16 & \equiv & 2 \pmod{7} \\ 5^2 & = & 25 & \equiv & 4 \pmod{7} \\ 6^2 & = & 36 & \equiv & 1 \pmod{7} \end{array}$$

Modular Arithmetik modulo einer Primzahl p ist ein Galoisfeld: $GF(p)$. Häufig wird auch Modular Arithmetik mit irreduziblen Polynomen betrieben. ($x^2 + 1$ ist ireduzibel, $x^3 + 2x^2 + x$ist es nicht, da $= x(x + 1)(x + 1)$.) Gern: $p(x) = x^n + x + 1$
Koeffizienten sind Integer modulo q, $p(x)$ hat Grad n: $GF(q^n)$.

Das asymmetrische public-key Verfahren RSA wurde 1977 entwickelt von Ron Rivest, Adi Shamir und Leonard Adleman. Der Public und der Private Key sind Funktionen eines Paares großer (mehrere hundert Stellen) Primzahlen. Um den Plaintext aus dem Public Key und dem Ciphertext zu ermitteln muß im Wesentlichen das Produkt der beiden Primzahlenzahlen faktorisiert werden, d.h. die Sicherheit von RSA beruht auf der Schwierigkeit der Faktorisierung großer Zahlen.
Das Verfahren basiert auf folgenden Set-up Schritten:

1. Um die Schlüssel zu erzeugen, wählen Sie zwei zufällig generierte Primzahlen p und q (möglichst gleich groß).
2. Berechnen Sie das Produkt $n = pq$.
3. Wählen Sie zufällig den Verschlüsselungs Key e, sodaß e und $(p-1)(q-1)$ relativ prim sind.
4. Berechnen sie den Entschlüsslungs Key d, sodaß $ed \equiv 1 \pmod{(p-1)(q-1)}$ (d ist also invers zu e in diesem Modul.)

RSA: Ver- und Entschlüsselung

e und n sind der Public Key, d ist der Private Key. p und q werden nicht mehr benötigt. Um eine Nachricht m zu verschlüsseln wird m in mehrere numerische Blöcke $m_i < n$ aufgeteilt.
Verschlüsselung: $c_i = m_i^e \bmod n$
Entschlüsselung: $m_i = c_i^d \bmod n$
Dabei ergibt sich folgendes **Problem**:
Ist $(m_i, n) = p$ oder q kann die Faktorisierung von n berechnet werden, indem im wesentlichen (m_i, n) berechnet wird. Dies kann allerdings vermieden werden, indem $(m_i, n) = 1$ für jeden Block überprüft wird. Ist $(m_i, n) \neq 1$, wird m_i nicht verwendet. Die Wahrscheinlichkeit, daß $(m_i, n) \neq 1$ ist, ist sehr klein für große p und q:

$$1 - \frac{\phi(n)}{n} = \frac{1}{p} + \frac{1}{q} - \frac{1}{pq}$$

HÜ7: Beweisen sie die Korrektheit der RSA Ver- und Entschlüsselungsformel für $(m_i, n) = 1$ und $(m_i, n) \neq 1$

Die Sicherheit von RSA liegt in der Schwierigkeit der Faktorisierung von n. Es ist allerdings nie bewiesen worden, daß dies die einzige Möglichkeit ist, um die Nachricht m aus c und e zu berechnen.

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

Der Angreifer kann d nicht berechnen, weil p und q nicht vorliegen, und damit der Modul $(p-1)(q-1)$ zur Berechnung von d nicht verfügbar ist. Weitere Möglichkeiten für Angriffe bestehen etwa in einem Rateversuch von $(p-1)(q-1)$ oder im Brute Force Angriff auf d. Beide Möglichkeiten sind allerdings nicht effizienter als eine Faktorisierung.

HÜ8: Warum ist RSA in der bisherigen Beschreibung (sog. Textbook RSA) nicht IND-CPA ? Wie wird mit RSA diese Sicherheitsstufe erreicht ?

Eve bekommt einen Ciphertext c, der mit RSA und dem Public Key von Alice verschlüsselt wurde. Eve will c entschlüsseln, also $m = c^d$ berechnen. Sie wählt $r < n$ zufällig und holt e. Dann berechnet sie

$$\begin{aligned} x &= r^e \bmod n \\ y &= xc \bmod n \quad x = r^e \bmod n \Rightarrow r = x^d \bmod n \\ t &= r^{-1} \bmod n \end{aligned}$$

Eve bringt Alice dazu, y mit ihrem Private Key zu unterschreiben und dabei y zu entschlüsseln. (Muß die Nachricht, nicht den Hash unterschreiben.) Alice schickt Eve dann $u = y^d \bmod n$

Eve rechnet

$tu \bmod n = r^{-1}y^d \bmod n = r^{-1}x^d c^d \bmod n = c^d \bmod n = m$, weil $x^d = r$ ist.

Trent hat die Aufgabe, Nachrichten mit RSA zu signieren. Mallory hat ein m', die er von Trent unterschrieben haben möchte, was Trent normalerweise nicht täte.
Mallory wählt ein x beliebig und berechnet $y = x^e \bmod n$, $m = ym' \bmod n$ und schickt m zu Trent. Trent unterschreibt und retourniert somit $m^d \bmod n$.

Mallory berechnet:

$$(m^d \bmod n)x^{-1} \bmod n$$
$$= ((x^e m')^d \bmod n)x^{-1} \bmod n$$
$$= x^{ed} x^{-1} \bmod n \cdot m'^d \bmod n$$
$$= m'^d \bmod n$$

Eve möchte, daß Alice m_3 unterschreibt. Sie generiert m_1 und m_2, sodaß $m_3 \equiv m_1 m_2 \pmod{n}$.
Wenn Alice m_1 und m_2 unterschreibt, kann Eve m_3 berechnen:
$m_3^d = (m_1^d \bmod n)(m_2^d \bmod n)$.

$\Longrightarrow$ Diese Szenen zeigen, daß RSA nie direkt benutzt werden darf, um von jemandem Fremden ein zufälliges Dokument zu unterzeichnen. Zumindest muß zuvor immer eine One-Way Hash Funktion benutzt werden!

$\Longrightarrow$ Eine wesentliche Frage ist, wie aufwändig es ist, die konstruierten und zu unterschreibenden Daten so hinzubekommen, dass diese tatsächlich etwas unterschreibenswertes darstellen !

Einige mögliche RSA Implementierung gibt jedem/r das gleiche n aber verschiedene e und d. Wird die gleiche Nachricht mit zwei relativ primen Exponenten verschlüsselt, kann der Plaintext ohne Private Key entschlüsselt werden.

$c_1 = m^{e_1} \bmod n$
$c_2 = m^{e_2} \bmod n$

Der/die FeindIn kennt n, e_1, e_2, c_1, c_2. $(e_1, e_2) = 1$
$\Rightarrow \exists r$ und s, sodaß $re_1 + se_2 = 1$, $r < 0$. Dann ist nur noch das folgende zu berechnen: c_1^{-1} und $(c_1^{-1})^{-r} \cdot c_2^s = m^{e_1 r} \cdot m^{e_2 s} = m \bmod n$.

Abhilfe: kein Sharing von n !

Wichtig ist eine gute Wahl von e für eine möglichst geringe Rechenzeit. Sehr oft wird 3, 7, $65537(2^{16} + 1)$ genommen, denn Zahlen mit wenig 1en in der binären Darstellung erlauben effiziente Berechnung.

Es gibt eine Attacke gegen RSA, wenn $\frac{e(e+1)}{2}$ linear abhängige Nachrichten mit dem gleichen e aber verschiedenen Public Keys verschlüsselt werden. Wenn die Nachrichten identisch sind, droht Gefahr bei der Verschlüsselung von e Nachrichten. Wenn die Anzahl der Nachrichten kleiner als e ist oder die Nachrichten keine Beziehungen haben, gibt es für diese Attacke keinen Angriffspunkt.

Abhilfe: Die Eliminierung dieser Sicherheitslücke ist sehr einfach. Es muß nur ein Padding der Plaintexte mit unabhängigen zufälligen Werten verwendet werden.

Sicherheit von RSA: Low Decription Exponent & Small Message Attacke

Bei rechenschwachen Empfangsgeräten kann auch die Wahl eines kleinen d von Vorteil sein. Ist $d \leq \frac{4}{3} n^{1/2}$ und $q < p < 2q$, kann d in linearer Zeit berechnet werden. Durch ein grosses e kann auch dieser Angriff vereitelt werden (und natürlich auch durch grosses d).

Wenn $c = m^e < n$, kann m mit der Komplexität des Ziehens einer n-ten Wurzel berechnet werden. Wenn ein kleines e aus Komplexitätsgründen notwendig ist, wird oft ein Padding bei den Plaintextblöcken angewandt, damit sie gross genug sind um diese Attacke zu vermeiden.

Sicherheit von RSA: Attacke gegen Verschlüsselung und Signierung

Alice verschlüsselt eine Nachricht mit Bobs Public Key und unterschreibt mit ihrem Private Key

$$(m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

Bob kann behaupten, Alice habe ihm m' und nicht m geschickt: Er kennt n_B und dessen Faktorisierung. Daher kann er den Diskreten Logarithmus bezüglich n_B berechnen. Er sucht ein x, sodaß $m'^x = m \bmod n_B$.

Wenn er nun xe_B als neuen Public Exponent publiziert und n_B behält, kann er behaupten Alice hätte ihm m' verschlüsselt mit xe_B geschickt.

Abhilfe: Sicherstellung eines fixen Verschlüsselungsexponenten.

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- DES
- RSA
  - Mathematische Grundlagen: Zahlentheorie
  - RSA Algorithmus
- Kecchak / SHA-3

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit

Angriffe gegen Hashfunktionen

- **Pre-image Angriffe**: auch Urbild Angriffe - aus einem Hash die unbekannte Nachricht zu finden (first pre-image attack) oder zu einer gegebenen Nachricht eine zweite Nachricht zu finden, die den gleichen Hash hat (second pre-image attack).
- **Kollisions Angriff**: finde zwei Dokumente, die auf den gleichen hashwert abbilden: $MD(m1) = MD(m2)$.
- **Kollisions Angriff mit chosen prefixes**: für zwei prefixes $p1$ und $p2$, finde zwei $m1$ und $m2$, sodass $MD(p1||m1) = MD(p2||m2)$.

MD-5 Angriff Timeline: 1996 erste Kollision der Kompressionsfunktion; 2004 Verfahren zur Berechnung von Kollisionen entwickelt (Wang et al.); erste praktische MD-5 Kollision mit zwei X.509 Zertifikaten; 2009 Attacke mit 200 Playstation 3 um wieder zwei Zertifikate anzugreifen, diesmal ist ein Certification Authority Zertifikat betroffen $\Longrightarrow$ Ende der Zertifizierung mit MD-5; Malware “Flame” wird 2012 entdeckt die MD-5 chosen-prefix collision Attack benutzt.

SHA-1 Angriff Timeline: 2005 erste Kollision in Kompressionsfunktion einer Runden-reduzierten Version; 2015 erste praktische Kollision der Kompressionsfunktion in 10 Tagen mit 64 NVIDIA GTX-970; 2017 SHAttered: erste Kollision des kompletten SHA-1 mit 6500 CPU-Jahren (2 PDF Daten bilden auf gleichen Hash ab), Zusammenarbeit Google & CWI Amsterdam.

$\Longrightarrow$ Panik in der Community, nur noch SHA-2 (SHA-224 – SHA-512) standen als sichere Hash Funktionen zur Verfügung, die von SHA-1 abgeleitet wurden, nachdem MD-5 und SHA-1 schrittweise gebrochen wurden.

$\Longrightarrow$ NIST initiiert eine öffentliche Ausschreibung / Wettbewerb mit Beginn 2007.

SHA-3 Auswahlprozess

1. Runde 1: Dezember 2008 - Juli 2009, mit Kandidaten Präsentationskonferenz und Festlegung der Bewertungskriterien; Bekanntgabe der 14 akzeptierten Kandidaten für Runde 2
2. Runde 2: Juli 2009 - Dezember 2010, mit Konferenz zur Kandidatendiskussion und Bekanntgabe der 5 Finalisten
3. Finalrunde: Dezember 2010 - Oktober 2012 mit Bekanntgabe des Gewinners
4. Offizielle SHA-3 Standardisierung August 2015.

Bewertungskriterien waren Sicherheit, Performance in Hard- und Software (auch mit Parallelisierung), Plattformübergreifende Implementierung, keine Patentverletzungen, einfach zu verstehen für Kryptoanalyse (auch bei schwächeren Varianten) und vollständig getestetes Design.

Das “Siegerverfahren” des SHA-3 Prozesses – Keccak – führt ein neues Paradigma in der Konstruktion von Hashfunktionen / symmetrischen Verschlüsselungsverfahren ein und beruht in keinster Weise auf den Vorgängerverfahren MD-5, SHA-1 und SHA-2. Wesentlich ist die sogenannte “Schwammkonstruktion” die dem Verfahren 2 Phasen vorgibt.

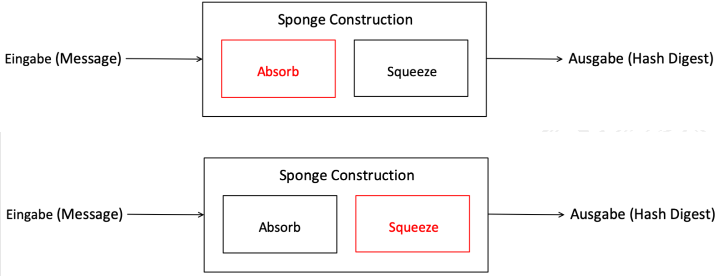


Figure: 2 Phasen von Keccak: Absorb & Squeeze.

Keccak: Sponge

Der Zustandsvektor S ist b bit gross, wird mit 0 initialisiert und teilt sich auf in die Bitrate r und die Kapazität c. Die Transformationsfunktion f basiert auf Permutationen und ist definiert als $f : \{0,1\}^b \Rightarrow \{0,1\}^b$.

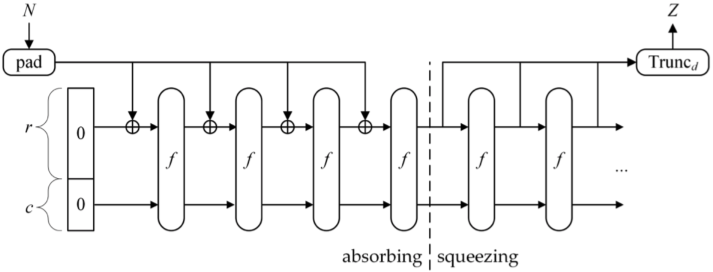
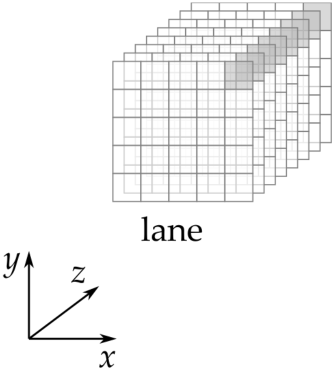


Figure: Schwammeigenschaft.

Im Absorbing werden Nachrichtenblöcke auf Länge r gepadded und mit dem Inhalt des Bitraten-Teils des Zustandsvektors ge-XORed. Anschliessend wird f auf den gesamten Zustandsvektor angewendet.

Keccak: Squeezing & Parameter

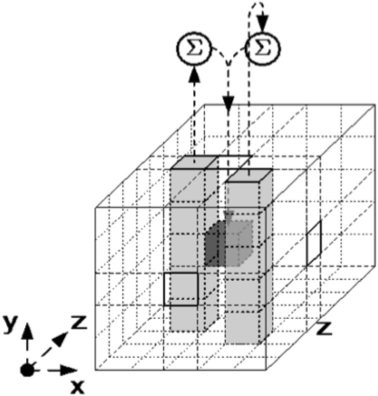
Im Squeezing wird ein Hashwert Z der Länge r dem Zustandsvektor entnommen; ist die Zielhashlänge d von Z kleiner als r, wird der Hashwert dahingehend gekürzt, ansonsten wird f auf den gesamten Zustandsvektor angewendet und anschliessend wieder ein Hashwert der Länge r dem Zustandsvektor entnommen der mit Z konkatiniert wird (u.s.w. bis d erreicht ist).



Der Zustandsvektor S besteht aus $b = 5 \times 5 \times 2^l$ Bits mit $0 \leq l < 7$ und wird angeordnet als 5×5 Bit Slices und 2^l Bits langen Lanes. Per default ist $r = 1024$, $c = 576$, $l = 6$, die Anzahl der Runden ist $12 + 2 * l$ (Anwendung von f auf den Zustandsvektor S) und die Hashlänge $d = \{224, 256, 384, 512\}$.

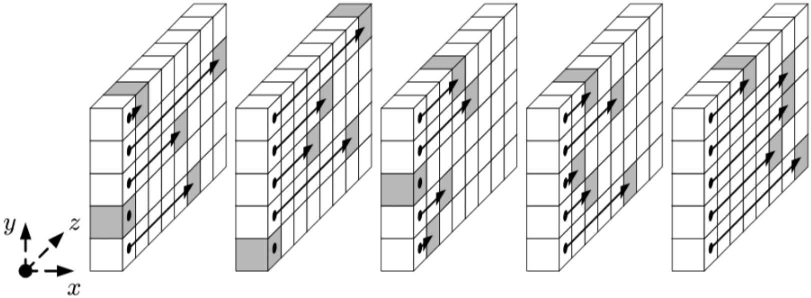
Keccak: die Rundenfunktion & θ

Die Rundenfunktion besteht aus einer Hintereinanderausführung von 5 unterschiedlichen Komponenten: $\theta, \rho, \pi, \chi, \iota$.

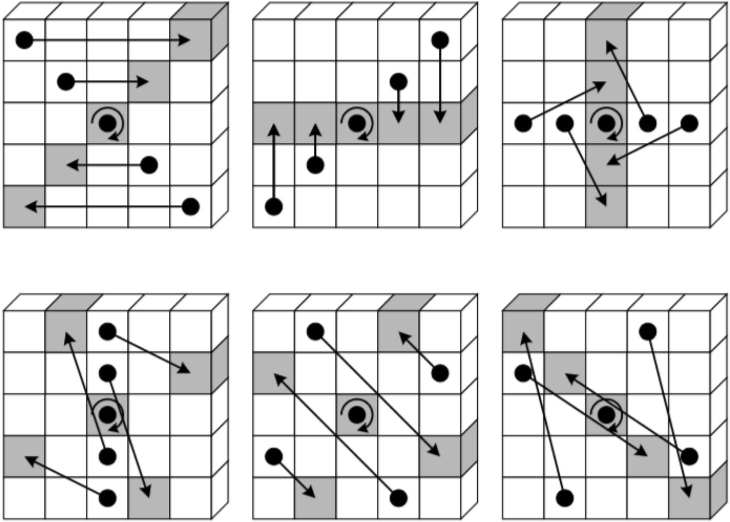


θ ist eine lineare Mischfunktion bei der die Wörter des Zustandsvektors (der Grösse 2^l) mit den Nachbarspalten geXORed werden, wobei die Wörter der rechten Nachbarspalte noch um 1 Bit links rotiert werden.

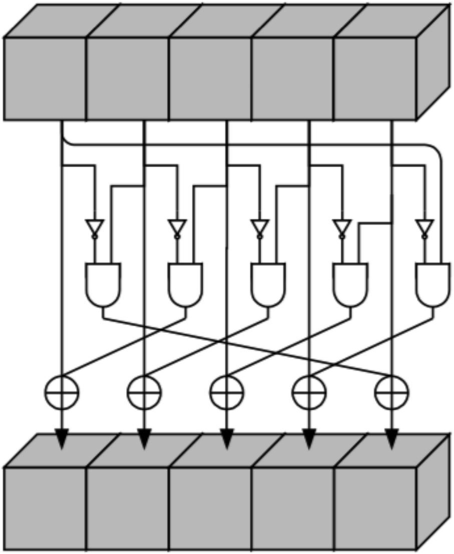
Die Rotationsfunktion ρ rotiert alle Wörter (bis auf das erste) um eine Konstante.



Bei der Permutationsfunktion π werden mit Ausnahme des ersten Wortes alle Wörter an eine neue Position von S verschoben.



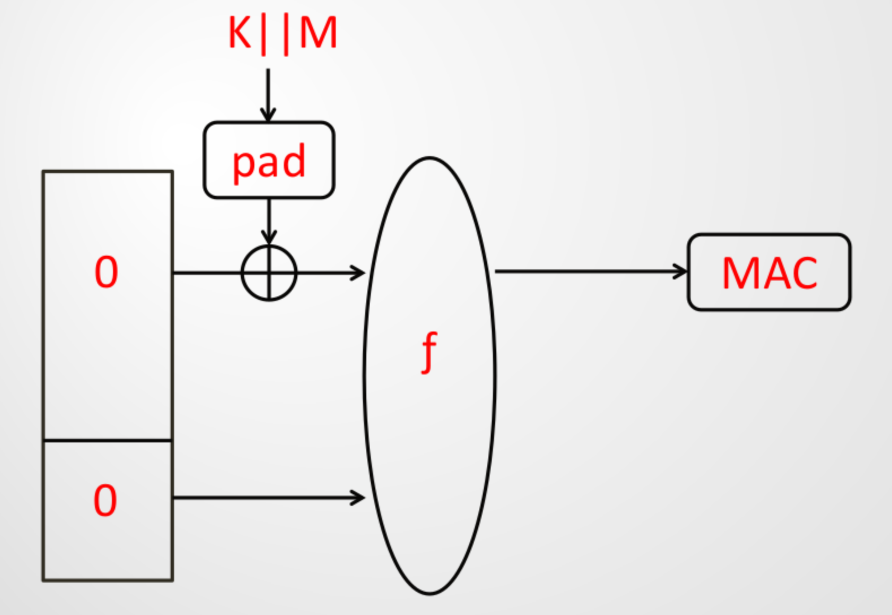
Keccak: χ & ι



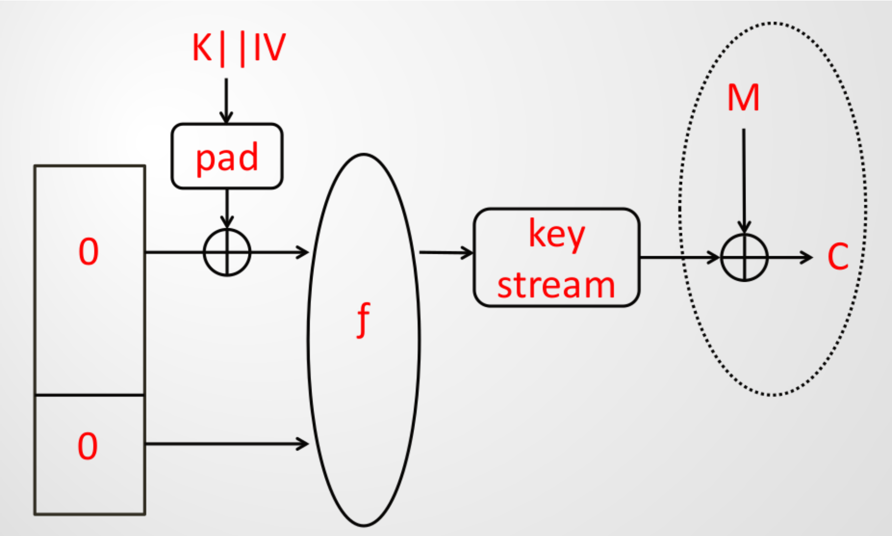
χ ist eine nicht-lineare Verknüpfung von 3 Wörtern des Zustandsvektors S die $\oplus$, logisches “nicht” und logisches “und” verwendet und auf jedes Wort angewendet wird.

ι XORed schliesslich zum ersten Wort des Zustandsvektors eine rundenabhängige Konstante.

Keccak ist flexibel: MAC



Keccak ist flexibel: Stream Cipher



Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen
- Stream Ciphers - Stromchiffren
- AES
- Weitere Public-Key Verschlüsselungs-Algorithmen
- Hashfunktionen aus Block Ciphers und MACs
- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen

Streamciphers: Arbeitsweise

Um streaming zu ermöglichen, wird für jedes incoming Bit ein Keybit für ein XOR bereitgestellt. Damit versuchen Streamcipher das Konzept des OTP anzuwenden, wobei der Schlüsselstrom eine wirklich zufällige Zufallsfolge nur approximieren kann. Durch die $\oplus$ Rechenregeln wird zur Entschlüsselung der identische Schlüsselstrom mit dem Ciphertext verXORed.

- Plaintext hat die Länge $|P| = k$, $P = P_1, P_2, \ldots, P_k$, mit P_i sind die Bits in P und $1 \leq i \leq k$.
- Schlüsselstrom wird erzeugt aus einem Schlüssel und dem Initialzustand des Streamciphers, $S = S_1, S_2, \ldots, S_k$.
- Ciphertext $C = C_1, C_2, \ldots, C_k$ wird berechnet durch $C_i = P_i \oplus S_i$ und $1 \leq i \leq k$.

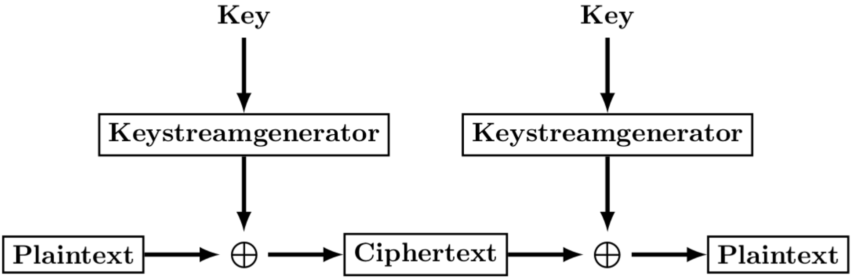
Ein Schlüsselstrom wird mit einem *kryptographischen* Zufallszahlengenerator erzeugt, der sich von Zufallszahlengeneratoren unterscheidet, die z.B. in Simulationsanwendungen verwendet werden.

1. **True random number generators**: Verwenden physische Quelle von Rauschen, die gesampled und nachbearbeitet wird (z.B. Zeitkomponenten im radioaktiven Zerfall, Latenzzeiten bei Festplattenzugriff, etc.). Es gibt hier keinen Schlüssel, d.h. die gesamte Folge ist der Schlüssel, keine Reproduzierbarkeit, schwierige Qualitätskontrolle.
2. **Pseudo / Quasi random number generators**: Zufallszahlen typischerweise rekursiv berechnet, S_0 ist der Seed (Key), $S_i = f(S_{i-1})$ wobei f sich auf auf deutlich mehrere Vorgängerwerte beziehen kann.
3. **Cryptographically secure PRNG**: seien $S_1, S_2, \ldots, S_k$ Bits des Keystreams bekannt, es ist computationally infeasible S_{k+1} zu bestimmen.

<u>**HÜ11**</u>: Was ist ein linearer Kongruenzgenerator (LCG) ? Illustrieren wie, warum ein LCG nicht kryptographisch sicher sein kann.

Synchrone Streamciphers

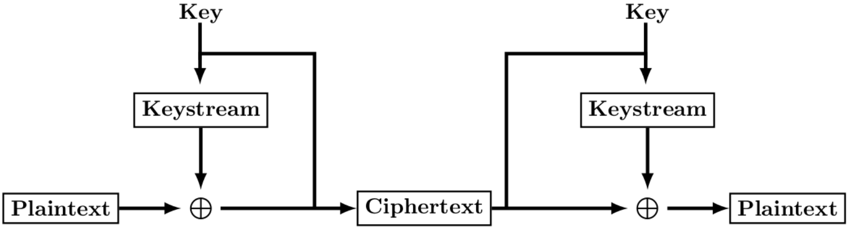
Ein synchroner Streamcipher generiert den Schlüsselstrom unabhängig von Plaintext und Ciphertext.



Recall: Blockcipher Betriebsmodi OFB und CTR sind synchrone Streamcipher.

Keystream kann vorberechnet werden, keine Fehlerfortpflanzung. Keine wiederholte Verwendung des Startzustands/Keys/Seeds ohne IV (siehe OTP), bei verlorenen Bits ist Synchronisation nötig.

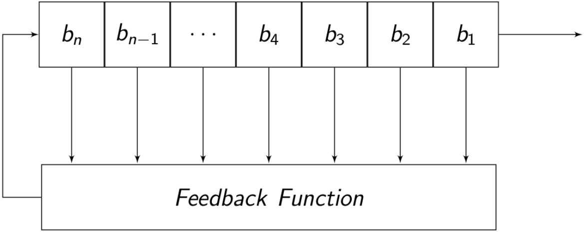
Im Gegensatz zu synchronen Streamcipher hängt bei einem asynchronen (selbstsynchronisierenden) Streamcipher der Schlüsselstrom von den vorhergehenden verschlüsselten Bits ab.



Recall: Blockcipher Betriebsmodi CFB (und CBC) sind asynchrone (selbstsynchronisierende) Streamcipher.

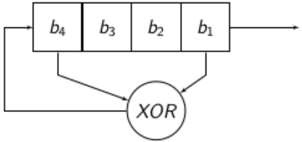
Verwendung des gleichen Startzustands unproblematisch, sicherer bei Plaintextredundanz (Bilder z.B.), selbstlimitierende Fehlerausbreitung;

FSR sind Basiskomponenten vieler Schlüsselstrom Generatoren, weil sie gut in Hardware realisiert werden können, weil sie Folgen mit guten statistischen Eigenschaften produzieren und gut mit algebraischen Methoden beschrieben/analysiert werden können.



Bestehen aus dem (n-bit) Schieberegister und einer Feedbackfunktion $f : \{0,1\}^n \longrightarrow \{0,1\}$; Bei Bitanforderung wird das LSB b_1 zurückgegeben, der Wert b_{n+1} als Ergebnis von f berechnet und die Werte b_{n+1} bis b_2 um eine Position nach rechts verschoben. Der Initialisierungszustand ist der Seed / Schlüssel.

Es gibt eine endliche Anzahl verschiedener Zustände ($2^n - 1$, 000... 00 ist nicht gestattet) und die Folgen werden periodisch. Ein hohes Ausmass an Gleichverteilung von binären Strukturen kann bei LSFR garantiert werden.

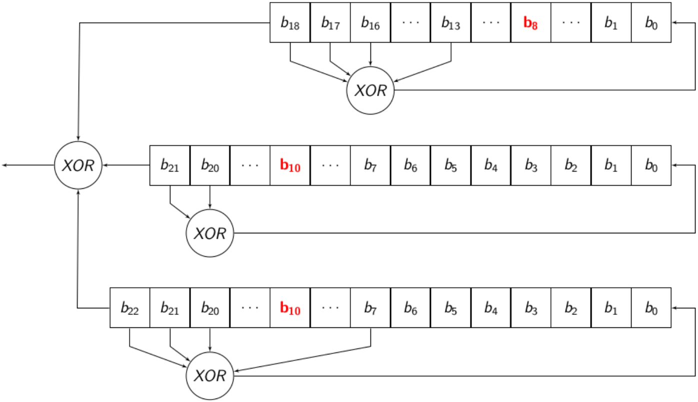


f ist ein XOR auf bestimmte Bits des Registers (die sog. “tap sequence”).

NFSR haben eine nicht-lineare Feedbackfunktion f, z.B. mit “und” bzw. Multiplikation und sind damit kryptoanalytisch deutlich resistenter als LFSR.

Allerdings kann eine Ausgaben Gleichverteilung nicht garantiert werden, die Periodenlänge variiert (oft vom Initialzustand abhängig) und die Hardwareimplementierung ist teurer. Trivium (aus dem eStream Portfolio, s.u.) ist ein Beispiel.

RC-4 (z.B. WEP, WPA, SSL, TLS, PDF, Skype) und DVD CSS unsicher/gebrochen. GSM A5/1 (Bild unten) erreicht nicht-Linearität durch unregelmässig getaktete LFSR, erreicht durch (rote) Clocking Bits (ist Bit $=$ Majority Bit, macht LSFR weiter, sonst stop), ebenfalls hochgradig unsicher.



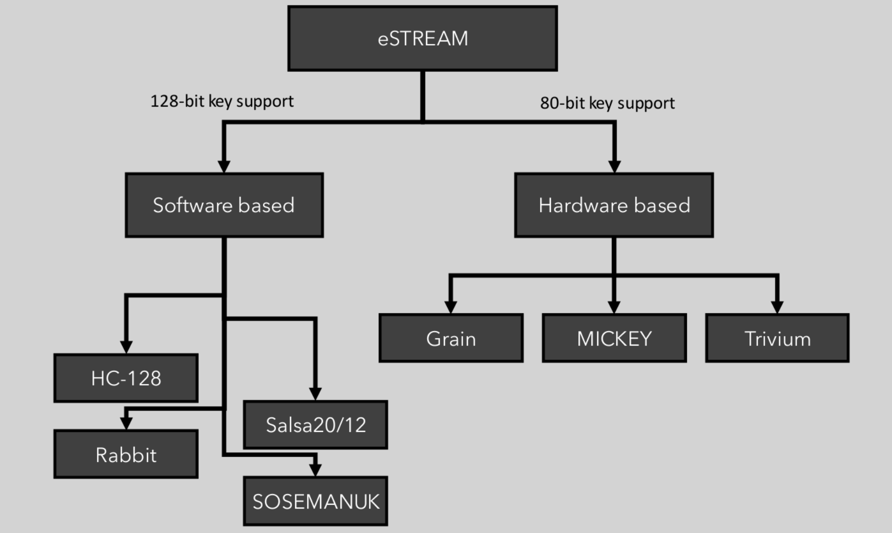
Das “e-Stream” Projekt

Europäisches Projekt (2004 - 2008), wurde initiiert, da beim Vorgängerprojekt NESSIE alle Stream-Cipher Finalisten bei der Kryptoanalyse durchgefallen sind. Aus 37 Kandidaten und 16 Finalisten wurden 7 effiziente und kompakte Streamciphers entwickelt in 2 Gruppen:

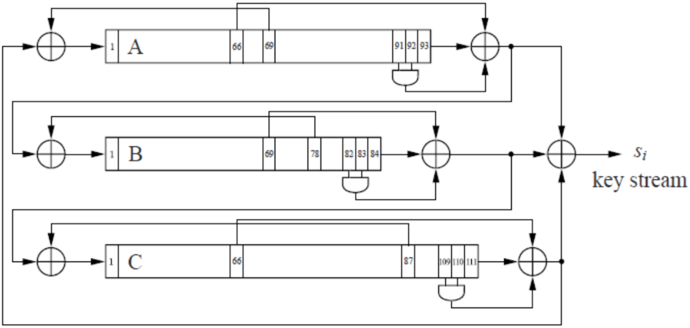
1. Profil 1: für Software-Implementierungen auf hohen Durchsatz optimiert, mit 128-bit Key Unterstützung.
2. Profil 2: für Hardware-Implementierungen optimiert, die mit eingeschränkten Ressourcen (gate count, RAM, Stromverbrauch) auskommen müssen, mit 80-bit Key Unterstützung.

Chacha baut auf Salsa20 auf und wird eingesetzt von Google in TLS, in OpenSSH, PRNG in Linux, als Standard vorgeschlagen für TLS, IPSEC/IKE.

e-Stream Portfolio



80 bit Schlüssel und 80 bit IV (gepadded mit 0 in A & B, C letzte drei Bit $= 1$), kombiniert drei NFSR (N durch Addition) über XOR mit 2^{64} bit Output; man wollte sehen wie weit man mit Vereinfachung gehen kann und trotzdem sicher bleiben. Bisher erstaunliche Resistenz gegen Kryptoanalyse, zukünftige Sicherheitsreserve eher dünn.



Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen
- Stream Ciphers - Stromchiffren
- AES
- Weitere Public-Key Verschlüsselungs-Algorithmen
- Hashfunktionen aus Block Ciphers und MACs
- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen

Weitere HÜs

HÜ12: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_07.pdf`, Aufgabe 5
HÜ13: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_07.pdf`, Aufgabe 6
HÜ14: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_07.pdf`, Aufgabe 8
HÜ15: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_07.pdf`, Aufgabe 15
HÜ16: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_08.pdf`, Aufgabe 13
HÜ17: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_08.pdf`, Aufgabe 21
HÜ18: `https://www.cosy.sbg.ac.at/~uhl/slides_ITsicher_09.pdf`, Aufgabe 17

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen
- Stream Ciphers - Stromchiffren
- AES
- Weitere Public-Key Verschlüsselungs-Algorithmen
- Hashfunktionen aus Block Ciphers und MACs
- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

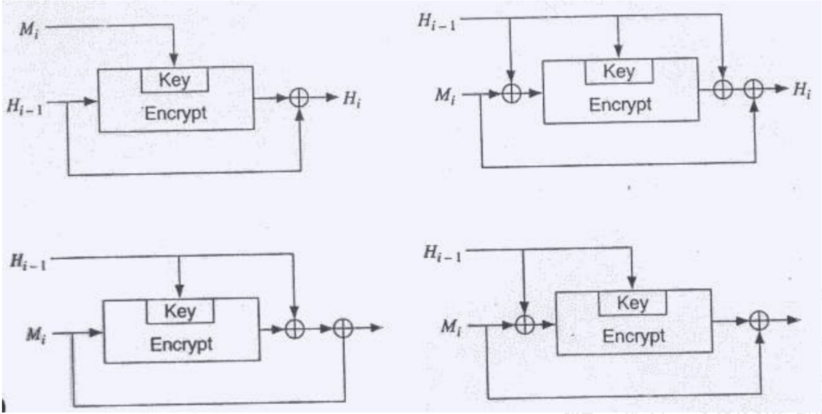
4 Weitere Kryptographische Algorithmen
- Stream Ciphers - Stromchiffren
- AES
- Weitere Public-Key Verschlüsselungs-Algorithmen
- Hashfunktionen aus Block Ciphers und MACs
- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen

Hashfunktionen aus Block Ciphers

Die Idee hinter dieser Hash Variante ist die Folgende:
Wenn der gewählte Block-Cipher sicher ist, müßte es auch die zugeordnete Hash Funktion sein.
Dieser Algorithmus könnte ganz offensichtlich ein CBC oder CFB mit fixem Schlüssel und Initialisierungsvektor sein. Der letzte Ciphertext Block ist der Hash Value. Dies ist aber nicht genug. Besser ist es, wenn die Nachricht als Schlüssel und der vorige Hash als Input verwendet wird. Somit entspricht die Blocklänge des Ciphers der Länge des Hashs.
Das tatsächliche angewendete Schema ist allerdings etwas komplizierter:

$$H_0 = I_H, H_i = E_A(B) \oplus C$$

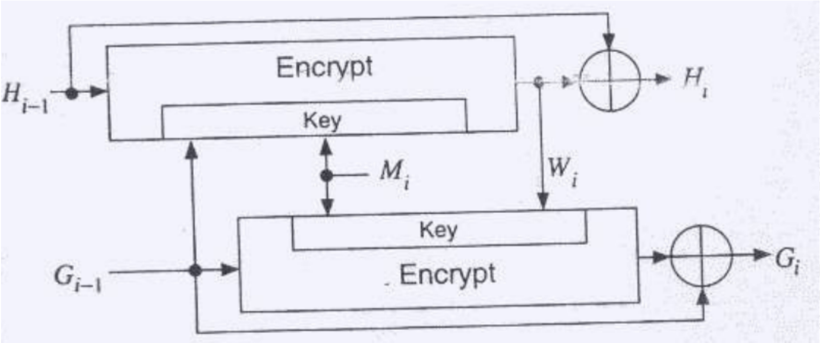
wobei I_H der Initialwert ist und A, B und C sind
$M_i, H_{i-1}, und(M_i \oplus H_{i-1})$ oder eine Konstante sein können.



Vier sichere Hash-Varianten bei denen Hash-länge gleich der Blockgrösse ist. Ist mit AES schon um unteren Ende der Hash-Sicherheit mit 128 Bit. Ist der Blockcipher in Hardware verfügbar, ist das Ganze unschlagbar schnell.

Hashfunktionen aus Block Ciphers: Tandem Davies Meyer

Ursprünglich mit dem 64-bit Blockcipher (Keylänge 128 Bit) IDEA entwickelt, ist diese Koppelung von zwei Blockciphern mit AES gut realisierbar, und zwar mit einer Keylänge von 256 Bit: der finale Hashwert wird erhalten durch $H_i||G_i$ und hat 256 Bit.



Messege Authentication Codes

MACs sind üblicherweise Weiterentwicklungen existierender kryptographischer Verfahren, als Grundlage werden Hashfunktionen (Bsp.: HMAC oder Keccak Variante) oder symmetrische Cipher (Bsp.: CBC-MAC) verwendet.

Es gibt mehrere Methoden um aus einer Hashfunktion einen MAC zu bauen: k ist der Key, H ist die Hashfunktion, M ist das pre-image und h der MACwert. Eine der einfachsten Methoden ist $h = H(k||M||k)$, die wegen theoretischer Schwächen eher gemieden wird.

Das **HMAC** Verfahren ist mehrfach standardisiert und gilt als sicher: Schlüssellänge und Hashlänge müssen identisch sein mit b Bits, die beiden Kostanten *opad* und *ipad* sind Bitfolgen der Länge b mit Wiederholungen von 01011100 bzw. 00110110.

$$h = H((k \oplus opad)||H(k \oplus ipad)||M))$$

Als Initialisierungsvektor wird meist eine Folge von 0-Bits verwendet, das Pre-image wird im CBC-Modus verschlüsselt, der letzte Block des Ciphertexts ist der MACwert. Das Pre-image muss gepadded werden damit die Blockgrösse stimmt. Das Verfahren ist heikel bzgl. Parameterwahl, unsicher bei Nachrichten unterschiedlicher Länge und sollte nicht uninformiert eingesetzt werden.

Counter Mode Cipher Block Chaining Message Authentication Code Protocol (CCMP, aus dem WLAN Bereich) kombiniert AES Verschlüsselung im CTR Mode und Authentifizierung mittels CBC-MAC und beruht auf dem CCM Modus (counter with CBC-MAC). Zuerst wird der CBC-MAC h berechnet, anschliessend werden M und h im CTR Modus verschlüsselt. Es stellt sich heraus, dass der gleiche AES Key für beide Vorgänge verwendet werden kann, wenn der IV im CBC-MAC nicht mit den Counter Werten kollidiert. Ein allgemeiner Sicherheitsbeweis dieser Kombination existiert, beruht auf der Sicherheit des Blockciphers.

Outline

1. Formalia
2. Einleitung
3. Klassische Kryptographische Algorithmen
   - Mathematische Grundlagen: Zahlentheorie
   - RSA Algorithmus
4. Weitere Kryptographische Algorithmen
   - Stream Ciphers - Stromchiffren
   - AES
   - Weitere Public-Key Verschlüsselungs-Algorithmen
   - Hashfunktionen aus Block Ciphers und MACs
   - Digital Signature Algorithmen (DSA) und Identifikations Schemata
   - Key-Exchange Algorithmen

Outline

1. Formalia
2. Einleitung
3. Klassische Kryptographische Algorithmen
   - Mathematische Grundlagen: Zahlentheorie
   - RSA Algorithmus
4. Weitere Kryptographische Algorithmen
   - Stream Ciphers - Stromchiffren
   - AES
   - Weitere Public-Key Verschlüsselungs-Algorithmen
   - Hashfunktionen aus Block Ciphers und MACs
   - Digital Signature Algorithmen (DSA) und Identifikations Schemata
   - **Key-Exchange Algorithmen**

Outline

1. Formalia

2. Einleitung

3. Klassische Kryptographische Algorithmen
   - Mathematische Grundlagen: Zahlentheorie
   - RSA Algorithmus

4. **Weitere Kryptographische Algorithmen**
   - Stream Ciphers - Stromchiffren
   - AES
   - Weitere Public-Key Verschlüsselungs-Algorithmen
   - Hashfunktionen aus Block Ciphers und MACs
   - Digital Signature Algorithmen (DSA) und Identifikations Schemata
   - Key-Exchange Algorithmen
   - Elliptic Curve Kryptographie Syst

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen
- Stream Ciphers - Stromchiffren
- AES
- Weitere Public-Key Verschlüsselungs-Algorithmen
- Hashfunktionen aus Block Ciphers und MACs
- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen

The **Paillier encryption** scheme allows two operations in the encrypted domain due to its additively homomorphic property. For any messages $m_1, m_2 \in \mathbb{Z}_n$:

$$\mathcal{D}_P(\mathcal{E}_P(m_1) \cdot \mathcal{E}_P(m_2) \bmod n^2) = m_1 + m_2 \bmod n$$

$$\mathcal{D}_P(\mathcal{E}_P(m_1)^{m_2} \bmod n^2) = m_1 \cdot m_2 \bmod n$$

The additively homomorphic property of the **Goldwasser-Micali encryption** scheme is for any $m_1, m_2 \in \{0, 1\}$ ($\oplus$ denoting *xor*)

$$\mathcal{D}_{GM}(\mathcal{E}_{GM}(m_1) \ \cdot \ \mathcal{E}_{GM}(m_2)) = m_1 \oplus m_2$$

In other words, if c_1 and c_2 are the encrypted values of m_1 and m_2, $(c_1 \cdot c_2) \bmod n$ will be an encryption of $m_1 \oplus m_2$.

For key generation, $\lambda = lcm(p-1, q-1)$ ($lcm =$ least common multiple) is computed and a random integer $g \in \mathbb{Z}^*_{n^2}$ is selected, resulting in the public/private key pair: pk_P: (n, g) and sk_P: (λ).
For encryption, we take a message m with $m \in \mathbb{Z}_n$ and select a random integer $r \in \mathbb{Z}_n$ (the latter provides semantic security and is not required for decryption).

$$c = g^m \cdot r^n \bmod n^2 \ .$$

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n,$$

where $L(u) = \frac{u-1}{n}$.

What about RSA ?

Recall that we have

$$c = m^e \bmod n \text{ and } m = c^d \bmod n \text{ with } e \cdot d = 1 \bmod (p-1)(q-1) .$$

Obivously, for messages m_1 and m_2 we get a multiplicative homomorphism since $m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e \bmod n$, thus,

$$\mathcal{D}_P(\mathcal{E}_P(m_1) \cdot \mathcal{E}_P(m_2) \bmod n) = m_1 \cdot m_2 \bmod n .$$

However, exactly this nice property is a problem in an adaptive chosen ciphertext attack (CCA) (RSA is said not to be *semantically* secure): To decrypt $c = m^e \bmod n$, compute $c' = c \cdot 2^e \bmod n$, decrypt c' by computing $((m \cdot 2)^e)^d$ which results in having m revealed.

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen
- Stream Ciphers - Stromchiffren
- AES
- Weitere Public-Key Verschlüsselungs-Algorithmen
- Hashfunktionen aus Block Ciphers und MACs
- Digital Signature Algorithmen (DSA) und Identifikations Schemata
- Key-Exchange Algorithmen

Grundprinzipien:

1. Quanten-Superpositionen: ein Elementarteilchen nimmt mehrere Zustände gleichzeitig an, die sich eigentlich ausschliessen, und der Zustand wird erst durch die Messung desselben festgelegt und die Superposition beendet.
2. Quanten-Verschränkungen: sind Zustände, die zwischen zwei oder mehreren Elementarteilchen auftreten können. Wird eine bestimmte Eigenschaft eines Teilchens gemessen, beeinflusst das Ergebnis eine Eigenschaft der anderen (mit ihm verschränkten) Teilchens.

Ein QuBit (QuantenBit) kann durch Superposition 0 und 1 gleichzeitig (mit bestimmten Wahrscheinlichkeiten) sein, erst beim Auslesen wird der Wert festgelegt. Ein Quantenregister ist eine Anzahl untereinander verschränkter QuBits, das sich bei n QuBits gleichzeitig in 2^n unterschiedlichen Zuständen befinden kann. Durch ein Quantengatter können logische Operationen auf 2^n Eingabewerte gleichzeitig ausgeführt werden, man kann allerdings nur ein Ergebnis auslesen.

Ein Quantencomputer ist nicht frei programmierbar, sondern arbeitet mit festverdrahteten Gattern, es handelt sich daher eher um "Quantenschaltkreise". Die grosse Stärke sind Anwendungen, bei denen aus zahlreichen ähnlich strukturierten Auswahlmöglichkeiten eine bestimmte gesucht wird (z.B. korrekter Primfaktor).

Der **Shor-Algorithmus** ist ein Quantencomputer Faktorisierungsverfahren für Primzahlprodukte (RSA !). Für dieses Verfahren braucht man 2 Quantenregister (Ein- und Ausgabe) von je 2^n QuBits, sodass die Bitanzahl der zu faktorisierenden Zahl zwischen 2^{n-1} und 2^n liegt. Da für die Berechnung eine Verschränkung zwischen den QuBits vorliegen muss, ist der technische Aufwand beträchtlich. Die # der benötigten Quantengatter zwischen Ein- und Ausgabe steigt kubisch mit der Anzahl der QuBits, die Laufzeit steigt in der vierten Potenz (klassisch wächst die Laufzeit stärker als jedes Ploynom in der Anzahl der Modul-Bits).

Da sich das diskrete Logarithmenproblem auf Faktorisierung zurückführen lässt, wären auch DSA, ElGamal u.v.a.m. betroffen.

Brute force Angriff mit dem Grover Algorithmus

Der **Grover-Algorithmus** ist ein Quantencomputer Verfahren zur Suche nach einem Eintrag in einer sortierten Liste. Man kann den Algorithmus aber auch zur vollständigen Schlüsselsuche bei einem symmetrischen Verschlüsselungsverfahren nutzen. Die zu durchsuchende Liste ist die Liste aller Schlüssel im Keyspace.

Durch Superposition können sehr viele Listeneinträge gespeichert werden, und man kann einen bestimmten Eintrag (z.B. den richtigen Schlüssel) finden, ohne jeden Eintrag abfragen zu müssen. Die Listeneinträge kann man sch im Quadrat angeordnet vorstellen.

Ein n-QuBit Register kann maximal 2^n Einträge verarbeiten, $2^{n/2}$ Operationen werden für die Identifikation der Lösung benötigt. Allerdings ist die Effizienz weit schwächer als beim Shor-Algorithmus – man muss durch das Grover Verfahren von einer Halbierung der Schlüsselbits ausgehen: z.B. ist damit AES mit 128 Keybits angreifbar.

Post-Quantum Cryptography

Asymmetrische Verfahren, die vermutlich nicht gegenüber Angriffen mit Quantencomputern anfällig sein. Nicht bewiesen, so sind es Verfahren gegen die es gegenwärtig keinen praktikablen Quantencomputer Angriff gibt:

1. Gitterbasierte Verfahren: NTRU (publiziert 1998) spielt als Einiges in der Praxis eine Rolle - verschlüsseln und signieren.
2. Hidden Field Equations (HFE): gut - hohe Verschlüsselungsgeschwindigkeit, geringe Grösse der Signaturen; schlecht - public keys mit über 100 KByte, nur Signatur.
3. Code-basierte Verfahren: das McEliece Verfahren (publiziert 1978) beruht auf Goppa-Codes (fehlerkorrigierender Code) - die Idee ist, z.B. in 2048 Bits (key) Fehler einzubetten, die für den Angreifer nicht zu korrigieren sind, aber für den Empfänger mit der (geheimen) Dekodierungs Matrix schon. schlecht - nur sicher mit Riesenschlüsseln (500 KByte - 4 Millionen Bit);
4. Hashbasierte Signaturverfahren: Signatur only; schlecht: lange Keys; gut: anspruchslose Implementierung;

In NTRU spielen Polynome der Form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

eine wichtige Rolle, mit a_i natürliche Zahlen die in Restklassensystmen manipuliert werden. Werden zwei Polynome multipliziert und das Ergebnis hat Grad n oder höher, wird eine "Reduktion" durch Teilung durch das Polynom $x^n - 1$ durchgeführt. Weiters brauchen wir noch 2 natürliche Zahlen p, q, z.B. $n = 503, p = 3, q = 256$.
Neben der Polynom Multiplikation gibt es offenbar auch eine Polynomteilung, in der das teilende Polynom $f(x)$ auch eine inverses Polynom $f^{-1}(x)$ besitzt mit $f(x)f^{-1}(x) = 1$.
Der Public-key $h(x)$ wird bestimmt durch: Alice wählt zwei Polynome $f(x)$ (invertierbar) und $g(x)$, und berechnet das Ploynom $h(x) = \frac{g(x)}{f(x)} \pmod q$. $f(x)$ ist der Private-key, wenn $h(x)$ und q gegeben sind, sind $g(x)$ und $f(x)$ eindeutig bestimmt (sonst gäbe es mehrere private keys). Das Berechnen von $h(x)$ gilt als one-way Funktion.

NTRU Ver- und Entschlüsselung

Bob benötigt für die Verschlüsselung den public-Key $h(x)$, die Nachricht wird als Polynom $m(x)$ kodiert, und zusätzlich wird ein Randomisierungsfaktor (“Blendpolynom”) $b(x)$ gewählt.

Verschlüsselung:

$$e(x) = pb(x)h(x) + m(x) \pmod{q}$$

Entschlüsselung:

1. Alice berechnet $a(x) = f(x)e(x) \pmod{q}$
2. Weiters ist $m(x) = \frac{a(x)}{f(x)} \pmod{p}$

Wie üblich, benötigt Alice für die Entschlüsselung $b(x)$ nicht. Die Verschlüsselungsfunktion gilt als Trapdoor one-way function. Der Beweis ist nicht ganz unkompliziert (da kommt das namensgebende “Gitter” Konzept vor).

Fazit: hochinteressantes Verfahren das die Zunkunft der public-Key Kryptographie sein könnte.

Outline

1 Formalia

2 Einleitung

3 Klassische Kryptographische Algorithmen
- Mathematische Grundlagen: Zahlentheorie
- RSA Algorithmus

4 Weitere Kryptographische Algorithmen

5 Netzwerksicherheit
- Sicherheit auf IP und DNS Ebene
- Sicherheit auf der Anwendungsebene

Outline

1. Formalia
2. Einleitung
3. Klassische Kryptographische Algorithmen
   - Mathematische Grundlagen: Zahlentheorie
   - RSA Algorithmus
4. Weitere Kryptographische Algorithmen
5. Netzwerksicherheit
   - Sicherheit auf IP und DNS Ebene
   - Sicherheit auf der Anwendungsebene