panikzettel.htwr-aachen.de

IT-Sicherheit

Version 7 | 24.10.2025

Tore Kunze Adrian Groh Jonas Schneider

Inhaltsverzeichnis

1 Einleitung

Vorwort

Sidenote: Warnung

Dieser Panikzettel ist noch nicht fertig. Die Zeit (und Motivation) fehlt für uns. Ich verspreche für den 2. Termin ist er fertig.

Dieser Panikzettel ist über die neue Vorlesung „IT-Sicherheit“, ein Pflichtfach des neuen Informatik Bachelors der RWTH Aachen.

Dieser Panikzettel ist Open Source auf `https://git.rwth-aachen.de/jonas.max.schneider/panikzettel` . Wir freuen uns über Anmerkungen und Verbessungsvorschläge (auch von offiziellen Quellen).

1.1 Ziele von IT-Sicherheit

IT-Sicherheit beinhaltet Vorbeugung, Erkennung und Verhinderung von Angriffen gegen eines der **CIA**-Ziele.

CIA

Confidentiality Nur autorisierte Personen haben **Zugriff** auf ein System
Integrity Nur autorisierte Personen können **Ressourcen** in einem System verändern
Availability Autorisierte Personen können **wie vorgesehen** auf Ressourcen in einem System **zugreifen**

Hierauf folgend kommen einige Beispiele für Angriffe auf eines (oder mehrere) der *CIA*-Ziele.

1.2 Angriffe gegen *Confidentiality*

Eavesdropping

Beim Transfer vom Daten werden diese abgehört/abgefangen.

Traffic Analysis

Zeiten und Adressinformationen werden genutzt, um herauszufinden, wer mit wem kommuniziert.

1.3 Angriffe gegen *Integrity*

Modification

Daten werden beim Transfer abfangen und modifiziert.

Masquerading

Die Source-Adressinformationen eines Datenpakets werden beim Transfer geändert.

Replay

Ein Datenpaket wird abgefangen und später (erneut) geschickt.

Repudiation

Eine Aktion wird geleugnet, z.B. ein spezifisches Datenpaket gesendet zu haben.

1.4 Angriffe gegen *Availability*

Denial of service

- Ein Webserver wird mit **Fake-Anfragen** geflooded
- Das Signal einer Verbindung wird mit einem **Störsignal** gejammed
- Das **Passwort** eines Nutzers wird **falsch eingegeben**, um den **Account** zu **blockieren**

Sonstiges

Es gibt grob zwei Prinzipien beim Design eines Kryptosystems (sei es ein Verschlüsselungsverfahren, ein Protokoll oder eine Anwendung).

Kreckhoffs Principle Ein Kryptosystem sollte sicher sein, selbst wenn alles über das System **außer der/die Schlüssel** bekannt ist.

Security by obscurity Das Gegenstück dazu. Sicherheit durch Geheimhaltung des Designs eines Kryptosystems. (Generell eine schlechtere Methodik, auch wenn sie natürlich kombiniert werden können.)

2 *Symmetric Encryption*

Bei *symmetric encryption* sind die ***encryption*** und ***decryption keys*** **identisch**.

2.1 *Caesar Cipher*

Ersetzt jeden verwendeten Buchstaben, durch den, der k Stellen im Alphabet danach kommt (k ist entsprechend der Schlüssel).

Sicherheit des *Caesar Cipher*

Der *Caesar Cipher* hat nur 25 verschiedene *Keys* $\{1, ..., 25\}$ und lässt sich somit **leicht mit *Bruteforce*** knacken. → **schlechte Verschlüsselungsmethode**

Bruteforce

Alle möglichen *keys* werden durchprobiert.

2.2 *Monoalphabetic Substitution Cipher*

Jeder **Buchstabe** wird durch einen anderen ersetzt gemäß einer **Ersetzungstabelle**.
Der *Caesar Cipher* ist insbesondere ein **Spezialfall** vom ***Monoalphabetic Substitution Cipher***.

Der key space entspricht nun allen Permutationen der Buchstaben.
→ ***Bruteforce*** ist **nicht mehr** gut möglich

Durch ***Frequency Analysis*** kann der *Cipher* leicht geknackt werden.

2.3 *Frequency Analysis*

Für jede Sprache gibt es **relative Häufigkeiten** der Buchstaben, was bei *Ciphern*, die die **Frequenzen** der Buchstaben **erhalten** insbesondere dazu führt, dass diese leicht entziffert werden können.

Monoalphabetic Substitution Ciphers erhalten insbesondere die Frequenzen und können daher mit *Frequency Analysis* geknackt werden.

2.4 *Perfect Secrecy* (Shannon)

Idee: Ein *ciphertext* soll **keine neue Information** über den *plaintext* preisgeben.

Definition: Perfect Secrecy

Ein *encryption scheme* hat ***perfect secrecy***, wenn für eine gegebene **Wahrscheinlichkeitsverteilung** Pr **auf dem *plaintext space*** $\mathcal{P}$ mit $\Pr(P) > 0$ für alle *plaintexts* P gilt:

Für jedes $P \in \mathcal{P}, C \in \mathcal{C}, K \in \mathcal{K}$ **zufällig gleichverteilt** ausgewählt, gilt $\Pr(P \mid C) = \Pr(P)$

Um dies zu erreichen muss der ***key space*** **mindestens so groß** sein wie der ***ciphertext space***, dieser wiederum muss **mindestens so groß** sein, wie der ***plaintext space***:

$|\mathcal{K}| \geq |\mathcal{C}| \geq |\mathcal{P}|$

Shannon's Theorem

Satz: Shannon

Sei $|\mathcal{P}| = |\mathcal{K}|$ und $\Pr(P) > 0$ für alle *plaintexts*.

Dann hat ein ***encryption scheme*** genau dann ***perfect secrecy***, wenn gilt:
1. Der *key* K wird **zufällig gleich verteilt** ausgewählt für jeden *plaintext* und
2. für jedes $P \in \mathcal{P}$ und $C \in \mathcal{C}$ gibt es **genau ein** $\boldsymbol{K \in \mathcal{K}}$ mit $E_K(P) = C$ (*plaintext* P wird veschlüsselt zu *ciphertext* C)

2.5 *One-Time-Pad* (OTP)

Wähle $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$ für ein $n \in \mathbb{N}$.

Key Generierung: Wähle **zufällig gleichverteilt** ein $K \in \mathcal{K}$ für jedes $P \in \mathcal{P}$, welches verschlüsselt werden soll

Verschlüsselung: $C = P \oplus K$

Entschlüsselung: $C \oplus K = P \oplus K \oplus K = P$

Satz: OTP

Der *One-Time-Pad* hat *perfect secrecy*.

Positives (+)	Negatives (-)
• **Einfach zu berechnen** ‣ Verschlüsselung und Entschlüsselung sind **dieselbe Operation** ‣ XOR ist eine **günstige** Rechenoperation • **So sicher wie theorethisch möglich** ‣ Wenn ein *ciphertext* gegeben ist, sind alle ***plaintexts* gleich wahrscheinlich**. ‣ Sicherheit **unabhängig** von den **Ressourcen** des Angreifers	• ***Key* muss so lang sein wie *plaintext*** ‣ **Unpraktisch** für die meisten Szenarien ‣ Trotzdem für diplomatischen und Geheimdienstverkehr benutzt • **Garantiert nicht *integrity*** ‣ Garantiert nur ***confidentiality*** ‣ Angreifer kann leicht den Text **verändern**, ohne dass dies erkannt wird • Offensichtlich **nicht** für alle Anwendungen **geeignet**

Moderne *encryption schemes* haben keine *perfect secrecy*, sind aber *computationally secure*.

Definition: Computational Security

Ein *encryption scheme* heißt **computationally secure**, wenn
- Alle bekannten Attacken gegen den *cipher* zu hohen Rechenaufwand haben, um ihn in absehbarer Zeit zu brechen

2.6 Attacken gegen Symmetric Encryption

Durch *Eavesdropping* hat ein Angreifer Zugriff auf den *ciphertext*.

Exhaustive Key Search / Brute-Force

Gehe alle möglichen *keys* ans dem *key space* durch.

Durchschnittlich muss er dafür $\frac{|K|}{2}$ *keys* ausprobieren.

Algebraic attacks

Wenn der Angreifer den *plaintext* kennt, kann er evtl. ein lineares Gleichungssystem mit den *key bits* als Unbekannte aufstellen und lösen.

2.7 Stream Ciphers

Bei *OTP* muss K immer genau die Länge von P haben.

Bei *Stream Ciphers* ersetzt man K durch einen *pseudo-random bit-generator (PRBG)*, welcher von einem *truly random key* K geseeded wird, und dann zusammen mit einem *initialization vector (IV)* für jedes K einen pseudo-random *key* der passenden Länge generiert.

Schwachstellen

Wenn *IV* für mehrere Verschlüsselungen verwendet wird, kann eine *known-plaintext attack* durchgeführt werden (**KRACK attack**).

2.8 Block Ciphers

Operieren auf *plaintext* Blöcken einer bestimmten Größe (*block length* $b \in \mathbb{N}$).

Plaintext space $\mathcal{P} = \{0,1\}^b$ und *ciphertext space* $\mathcal{C} = \{0,1\}^b$

Brauchen typischerweise einen *mode of encryption*, der spezifiziert, wie ein *plaintext* mit einer Länge über b verschlüsselt wird.

2.9 AES

Der Input läuft dann durch mehrere Runden. In jeder Runde gibt es die Operationen *Substitute Byte (SB)*, *Round Key Addition (KA)*, *Shift Row (SR)* und *Mix Column (MC)*.

Ich denke nicht, dass man die genaue Funktionsweise für die Klausur auswendig können muss.

2.10 Modes of Encryption

Electronic Codebook Mode (ECB)

Teilt einfach den Input in Blöcke der richtigen Länge und fügt falls Nötig am Ende noch ein Padding hinzu.

Problem: Die selben *plaintext* Blöcke ergeben denselben *ciphertext*, wodurch sich Muster aus dem *plaintext* im *ciphertext* wiederfinden lassen. ⇒ sollte nicht benutzt werden

Cipher Block Chaining Mode (CBC)

Ähnlich wie ECB, nur dass jeder Block vor der verschlüsselung noch mit dem *ciphertext* des vorigen Blocks $\oplus$ gerechnet wird (und der erste Block mit einem öffentlichen IV).

Man sollte für jeden *plaintext* einen neuen IV verwenden, da ein Angreifer ansonsten erkennen kann, ob mehrfach derselbe *plaintext* verschlüsselt wurde.

Ist anfällig gegen *padding-oracle attacks*. ⇒ sollte auch nicht mehr benutzt werden

Counter Mode (CTR)

Bei CTR wird auf den E_K vom ersten Block $\mathrm{IV}+1$ auf den zweiten Block $\mathrm{IV}+2$ usw. angewendet und dann unabhängig von einander jeder Block mit dem jeweiligen E_K $\oplus$ gerechnet.

Dadurch ist kein Padding notwendig, und Ver- und Entschlüsselung kann parallelisiert werden.

CTR macht also aus einem *block cipher* einen *stream cipher*.

Die abgewandelte Variante des Galois Counter Mode (GCM) wird in Abschnitt 3.4 behandelt.

3 Symmetric Integrity Protection

Modification Detection Code (MDC) werden über einen **secure** channel gesendet (die Nachricht nicht). Dann wird die MDC-Funktion auf die Nachricht angewand und geguckt ob MDC gleich ist.

Message Authentication Code (MAC) werden mit der Nachricht über den unsicheren Channel gesendet. Um eine MAC zu erstellen (und zu überprüfen) benötigt man den Key.

3.1 Hash functions

Definition: Hash Funktion

Eine Hash Funktion h muss folgende Eigenschaften aufweisen:
1. *compression*: h mapped eine Eingabe unbestimmter Länge auf eine **fixe** Länge.
2. *ease of computation*: Die hash funktion muss effizient berechenbar sein.

Aus der ersten Eigenschaft ergibt sich natürlich, dass es (unendlich viele) Kollisionen geben muss.

Definition: Cryptographic Hash function

preimage resistance Gegeben einem $y = h(x)$, aber nicht x. Es ist nicht möglich x' zu finden mit $h(x') = y$. Es kann auch $x = x'$ gelten.
second preimage resistance Gegeben einem $x, h(x)$. Es ist nicht möglich ein $x' \neq x$ zu finden mit $h(x') = h(x)$.
collision resistance Es ist nicht möglich ein x, x' zu finden mit $h(x') = h(x)$.
[1]

Satz: Geltende Implikationen

collision resistant $\implies$ *second preimage resistant.*

Alle anderen Implikationen gelten nicht.

Eine *cryptographic hash function* ist nun eine die alle 3 Eigenschaften erfüllt.

3.2 HMAC

MAC auf Grundlage einer *cryptographic hashs*

$$\text{HMAC}_K(M) = h(K \oplus \text{opad} \parallel h(K \oplus \text{ipad} \parallel M)) \qquad 2$$

ipad und *opad* sind festgelegt und unterscheiden sich

3.3 CMAC

MAC mittels Symmetrischer Block Verschlüsselung.

$M = M_1 \parallel M_2 \parallel ... \parallel M_n$

1. Fall $\|M_n\| = b$ M_n passt genau in einen Block.

   $\text{CMAC}(M) = K_{\mathbf{1}} \oplus M_n \oplus E_k(M_{n-1} \oplus E_k(... \oplus E_k(M_1)))$

[1] Alle „nicht möglich“ sind natürlich nur *computationally infeasible*
[2] $\oplus$ kommt immer zuerst

2. Fall padde $M'_n = M_n 1 \underbrace{0...0}_{b - \|M_n\| - 1}$

$$\text{CMAC}(M) = K_{\mathbf{2}} \oplus M'_n \oplus E_k(M_{n-1} \oplus E_k(... \oplus E_k(M_1)))$$

K_1, K_2 generieren wir uns aus K, aber benötigen unterscheidliche, da sonst $\text{CMAC}(M) = \text{CMAC}(M')$ wäre wenn $M' = M10...0$

Replay Protection in MAC & Verschlüsselungsreihenfolge

Um gegen einfach kopieren der Datei mit MAC zu verhindern müssen wir Sachen (wie z.B. timestamp, RNG, SQN) einbauen. Bei RNG muss Bob zuerst die Zufallszahl senden, die dann mit der $\text{MAC}_K(M \parallel \text{RAND})$ mitgeschickt wird.

Man sollte immer erst Verschlüsseln und dann den MAC berechnen, da dann MAC keine sicherheitsrelevanten Informationen enthällt und man MAC erst überprüfen kann, bevor dem aufwändigeren Entschlüsseln.

3.4 GCM

Der GCM Encryption Mode kann beides gleichzeitig.

Wir haben dabei $A_1 \parallel ... \parallel A_m \parallel P_1 \parallel ... \parallel P_n$ $m < n$. Wir wollen A_i integrity protecten und P_i verschlüsseln und integrity protecten.

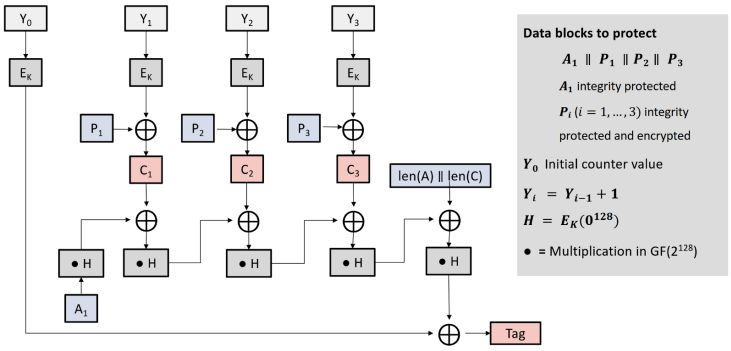

Abbildung 1: Operationsdiagramm GCM[3]

4 Asymmetric Encryption & Signatures

Alles hier beruht darauf einen *public key* p_k zu haben den jeder kennt (man geht auch davon aus, dass jeden ihn kennt). Und einen *private key* s_k den Niemand außer man selber kennt.

4.1 RSA

[3]Hier wird nur A_1 dargestellt, aber bei mehreren A_i werden die nacher äquivalent mit eingebunden