Version 2023b

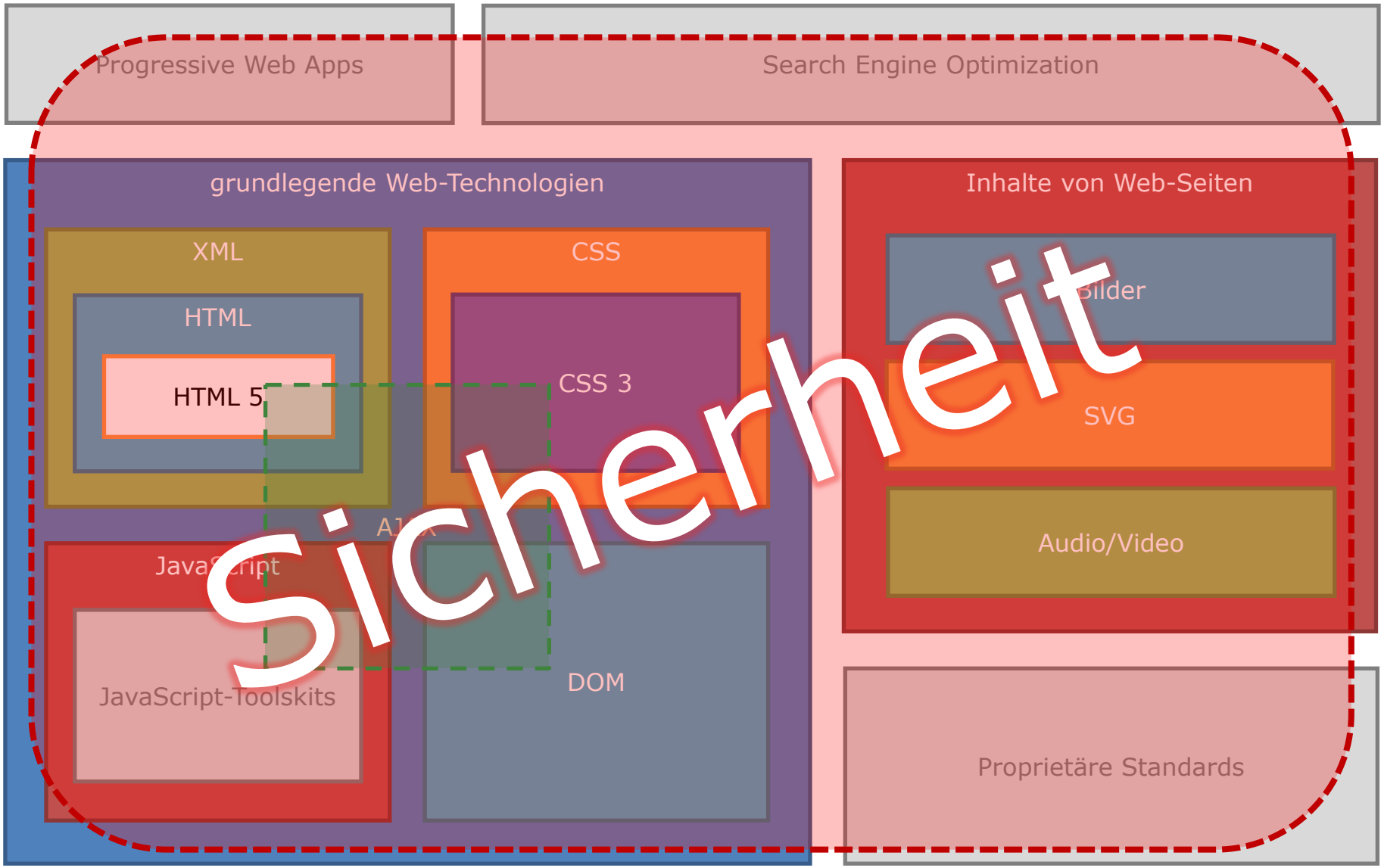
BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Grundlagen der Web-Programmierung

Exkurs: Sicherheit

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung



Motivation

Kommunikation muss bewusst gewählten Eigenschaften genügen

- Integrität und Konsistenz
- Verfügbarkeit und Fairness
- Anonymität und Obskurität
- Zurechenbarkeit und Nachweisbarkeit

Gefahren und Schutzziele

Gefahren

- nicht feststellbar mit irreversiblem Effekt aber vermeidbar
  - unautorisierter Zugriff auf Informationen
- unvermeidbar aber feststellbar und mit reversiblem Effekt
  - unautorisierte Veränderung von Informationen
  - unautorisierte Vorenthaltung von Informationen

klassische Schutzziele – Definition

- Vertraulichkeit nur autorisierte Nutzende erhalten Informationen
- Integrität Information sind korrekt, vollständig und aktuell (oder dies ist nachweisbar nicht der Fall)
- Verfügbarkeit Information sind zugreifbar wann und wo autorisierte Nutzende sie benötigen

→ es muss klar sein, wer in welcher Situation autorisiert ist

→ Informationen sind: Daten, Programme, Hardware-Strukturen, …

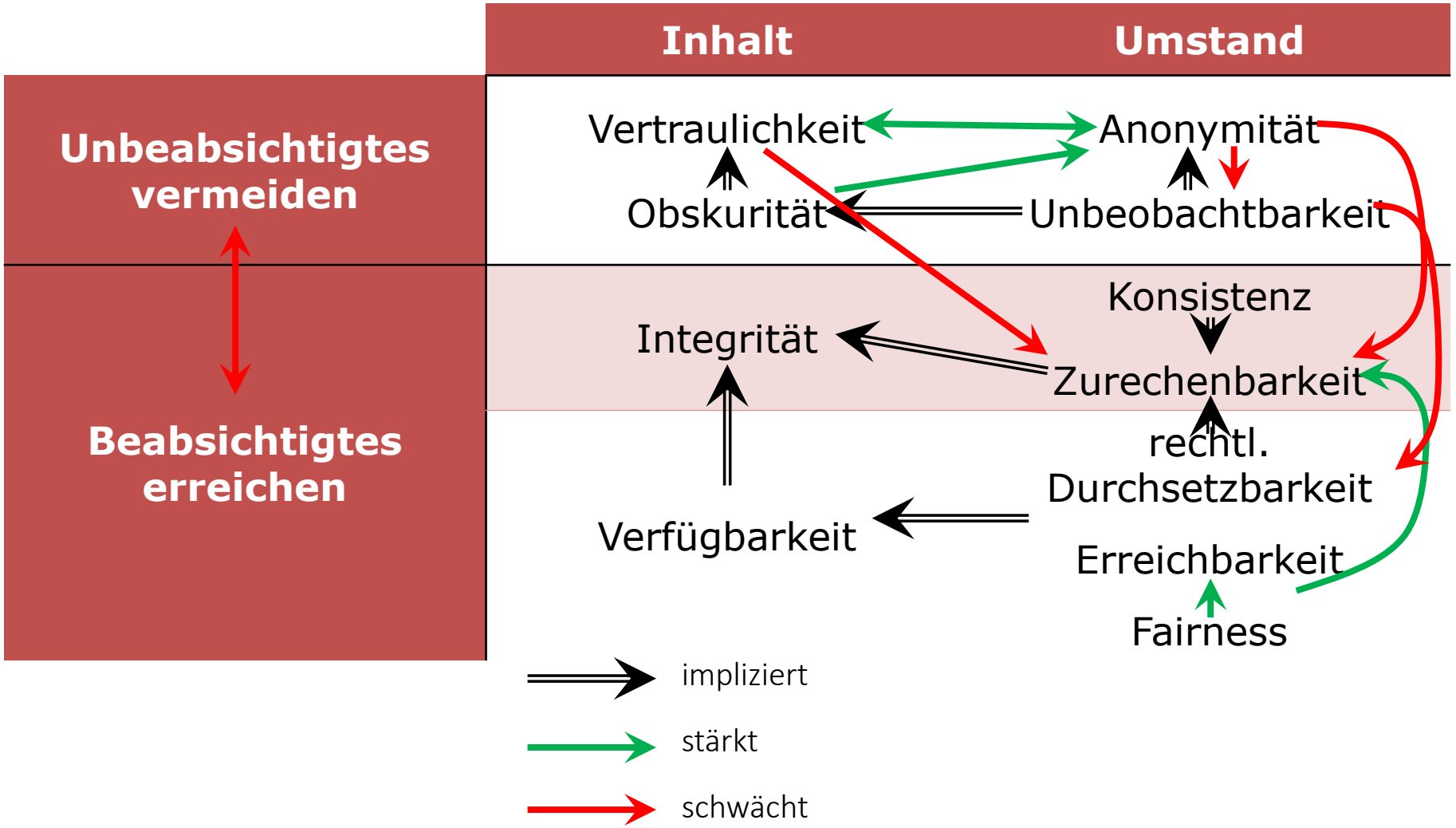
→ kann sich nur auf Systeminterna beziehen

⇒ Achtung: Angreifende mit hinreichend Zeit und Geld knacken alles!

klassische Schutzziele – Mittel

- Vertraulichkeit
  - vertrauliche Information — Ende-zu-Ende-Verschlüsselung
  - anonyme Sender/Empfänger — Obfuskation von Verkehrsdaten
- Integrität
  - Erkennen von Veränderungen — Authentizitätssystem
  - nachweisbarer Versand — Nachrichtensignatur
  - nachweisbarer Empfang — Quittung
  - Identitätsnachweis — Nachrichtensignatur
- Verfügbarkeit
  - Ausfallsicherheit und Redundanz — diverse Netzwerke; faire Teilen von Ressourcen

erweiterte Schutzziele

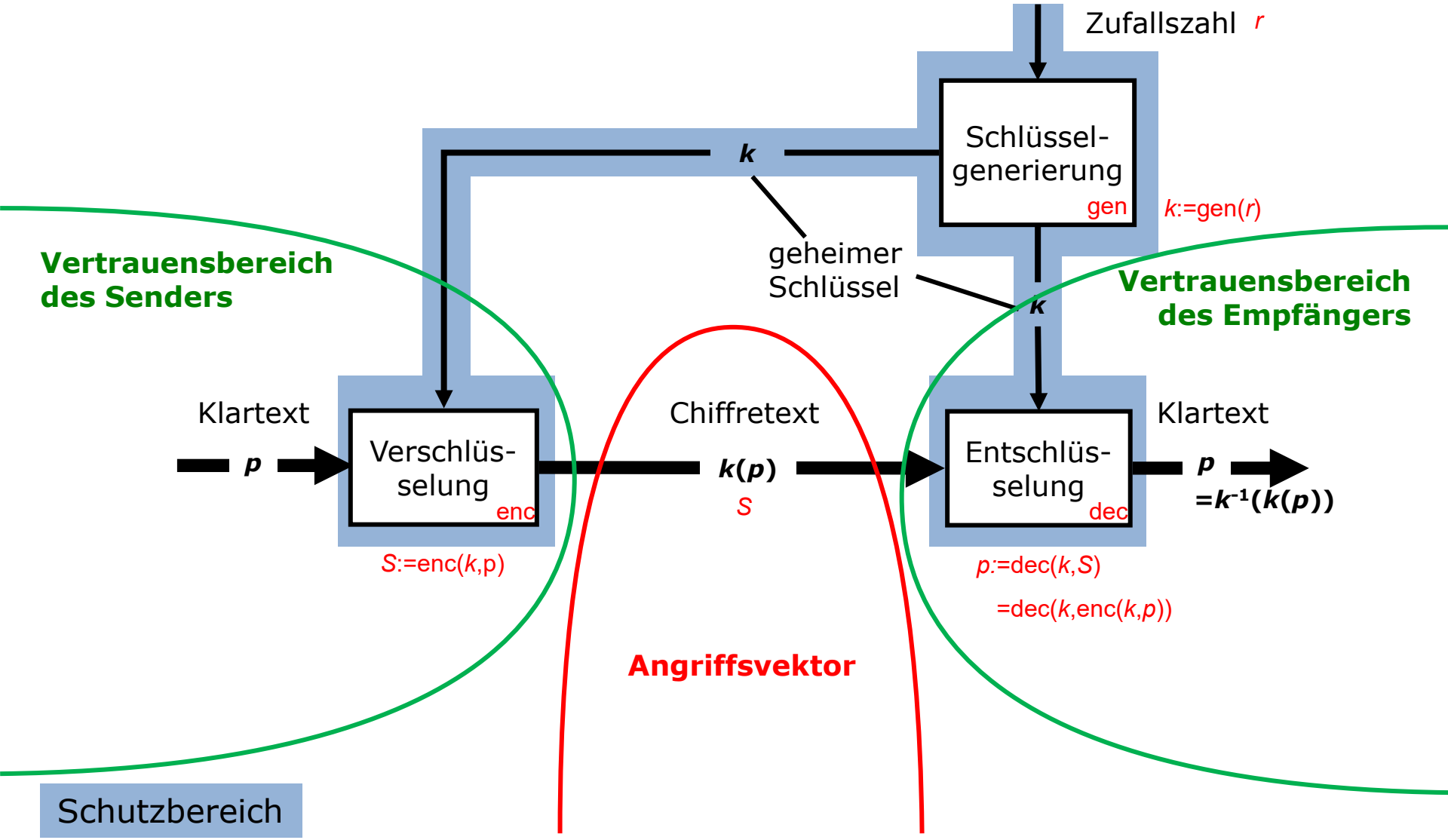


symmetrische Kryptographie

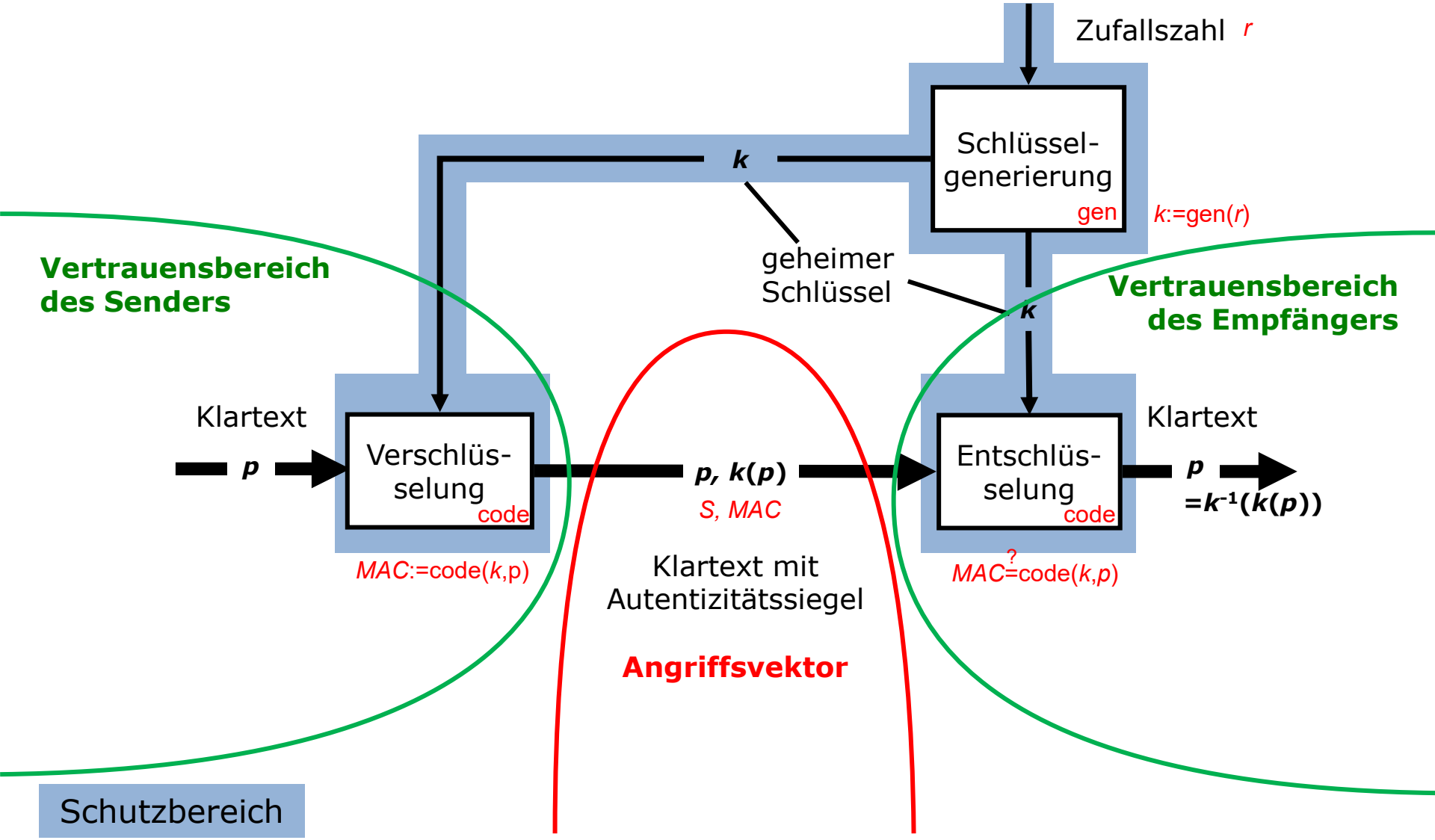
Eigenschaften symmetrischer Kryptographie

- schnelle Berechenbarkeit
- Sicherheitsniveau wird bestimmt durch
  - verwendetem Algorithmus (Cipher)
  - Schlüssellänge
- Kombination von Schlüsseln und/oder Ciphern möglich (Stacking)
  → vergleichbar Bankschließfach: Kundenschlüssel + Bankschlüssel

Verschlüsselung (undurchsichtige Box mit 2 identischen Schlüsseln)



Authentifizierung (transparente Box mit 2 identischen Schlüsseln)



asymmetrische Kryptographie

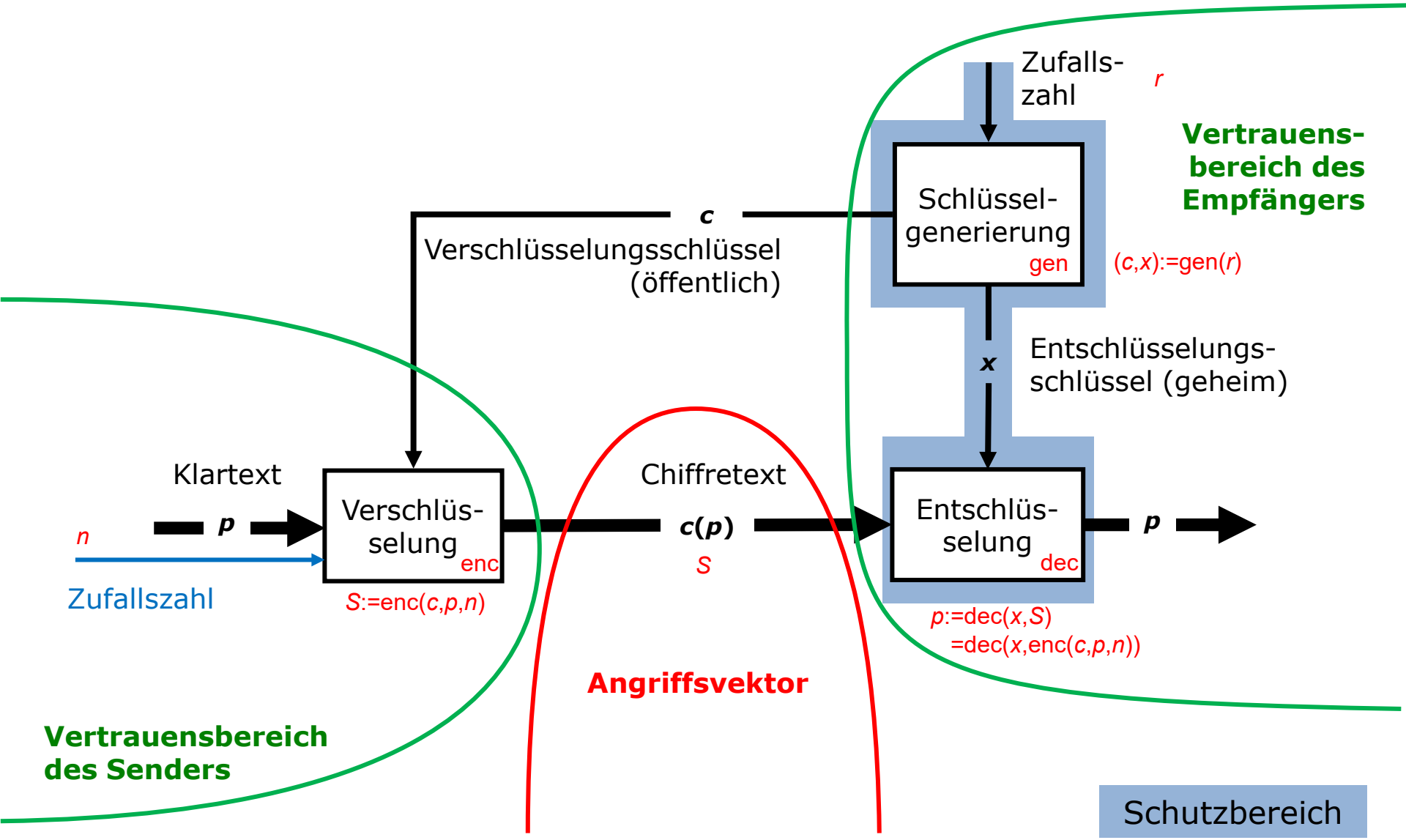
Eigenschaften asymmetrischer Kryptographie

- Berechenbarkeit ist komplex (hart und damit langsam)
- Sicherheitsniveau wird bestimmt durch
  - mathematische Grundlage
  - verwendetem Algorithmus
  - Schlüssellänge

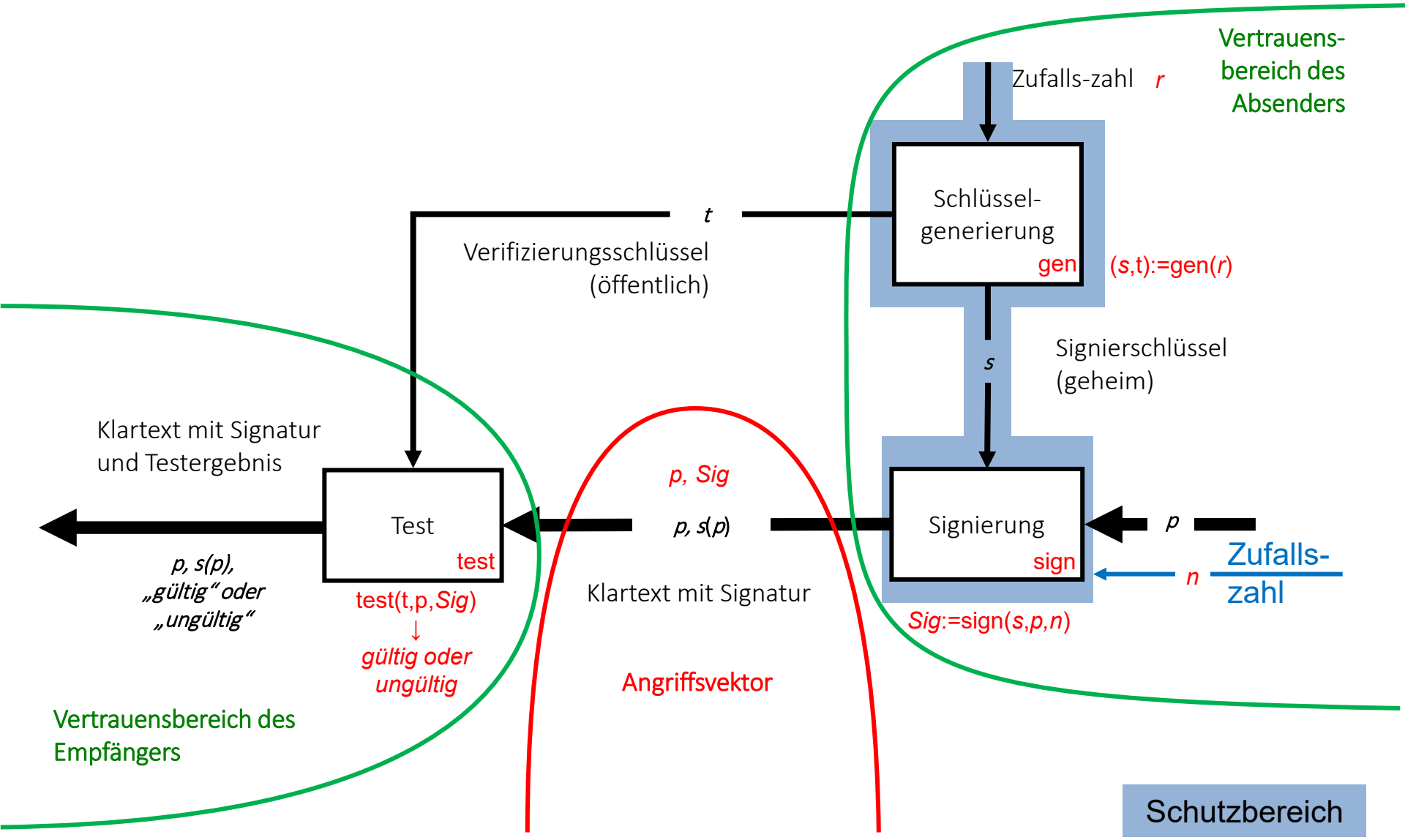
asymmetrische Cipher-Funktionen

- müssen sein
  - einfach berechenbar wenn alle Parameter bekannt sind
  - super-harte Probleme wenn (bereits einzelne) Parameter unbekannt
- Beispiele
  - auf elliptischen Kurven definierte geometrische Addition
    → super-hartes Problem: finde korrekte Zahlen eines Punktes, der selbst gefunden werden muss auf einer noch zu bestimmenden Kurve
  - Primfaktorzerlegung
    → super-hartes Problem: finde zwei sehr große Primzahlen so, dass
    $(x \equiv y \bmod z)$
    mit $(x \equiv y \bmod a)$ und $(x \equiv y \bmod b)$
    und $(z = a \cdot b)$
    während a und b prim

Verschlüsselung (undurchsichtige Box mit Schnappschloss und 1 Schlüssel (beim Empfänger))



Authentifizierung (transparente Box mit 1 Schlüssel (beim Absender))



Performanzverbessrung: hybride Verschlüsselung

durch Kombinieren asymmetrischer mit symmetrischer Verschlüsselung

- Sicherheitsvorteile asymmetrischer Verschlüsselung
- Geschwindigkeit symmetrischer Verschlüsselung

Grundprinzip

1. generiere symmetrischen Verschlüsselungsschlüssel S
2. verschlüssele und signiere S asymmetrisch
3. sende verschlüsselten und signierten S an den Kommunikationspartner
4. verschlüssele alle Daten mit S
5. generiere regelmäßig einen neuen S

Verfahren nach Rivest, Shamir und Adleman

Rivest-Shamir-Adleman (RSA)

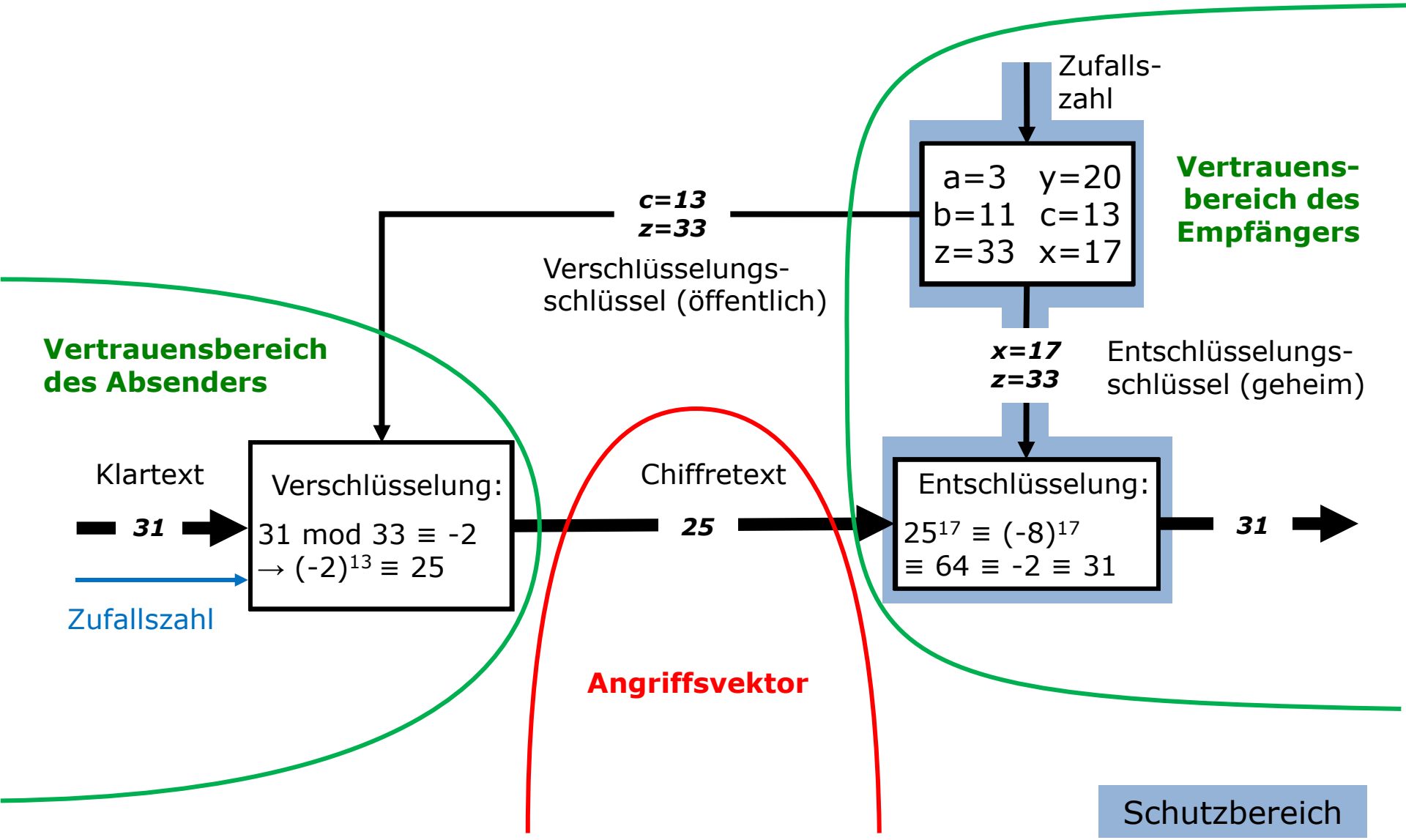
1. Wähle zufällig zwei sehr große Primzahlen a und b
2. berechne $z = a \cdot b$ sowie $y = (a-1) \cdot (b-1)$
3. wähle x (geheimer Schlüssel) mit $3 \leq x \leq y$ und $\mathrm{ggT}(y, x) = 1$
4. berechne c aus a, b, x als multiplikatives Inverses von $x \bmod y$
   $\rightarrow c \cdot x \equiv 1 \bmod y$
5. veröffentliche c (öffentlicher Schlüssel) und z (Operationsmenge)

   die Zahlen ergeben: $(p^c)^x \equiv p^{c \cdot x} \equiv p^{x \cdot c} \equiv (p^x)^c \equiv p \bmod z$

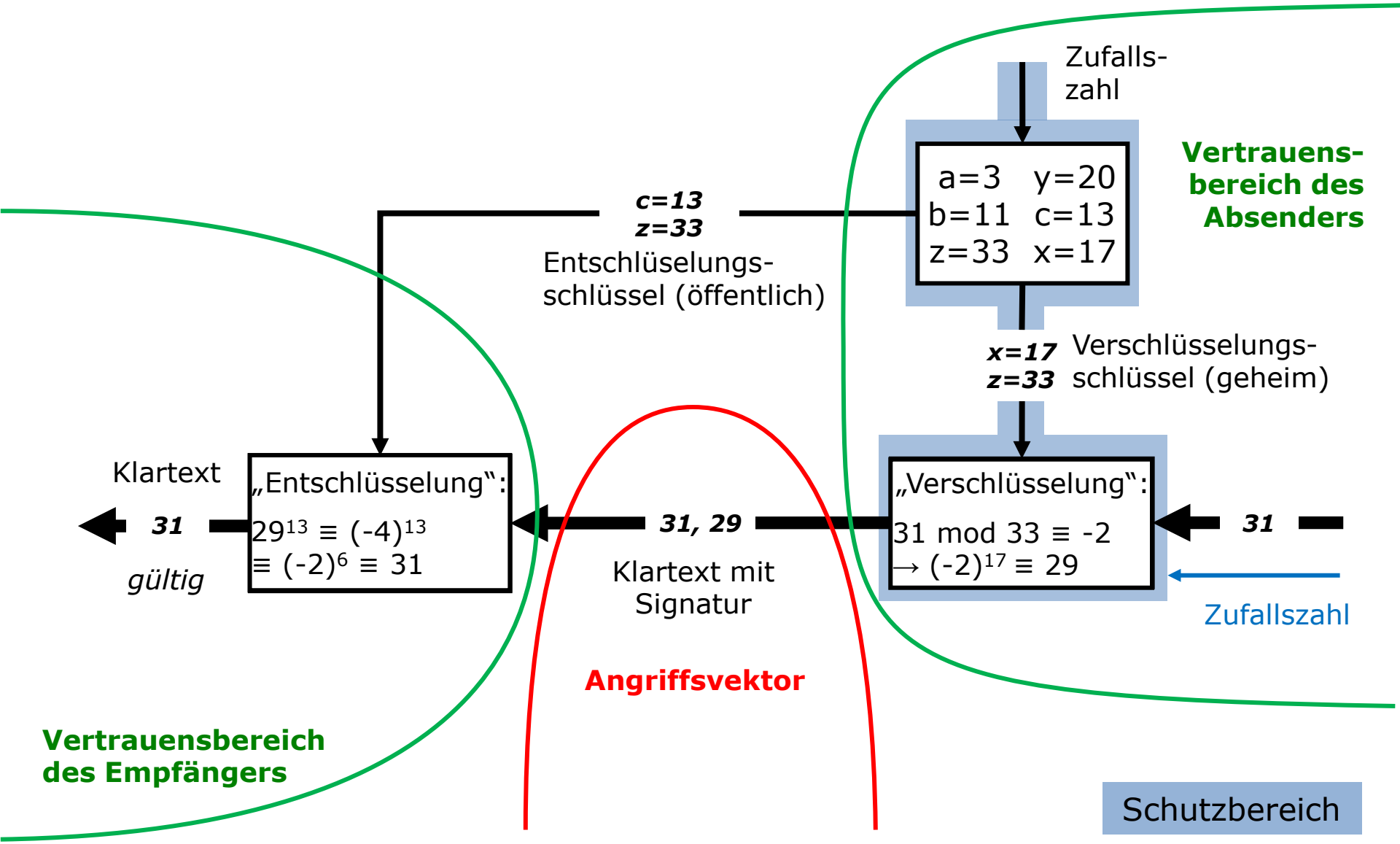
öffentlicher Schlüssel kann verwendet werden

- zum verschlüsselten Senden an den Schlüsselerzeuger
- zum Verifizieren einer Signatur des Schlüsselerzeugers

naive Verschlüsselung mit RSA



naive Authentifikation mit RSA



Härten von RSA mittels Hash-Funktion

- naives RSA kann einfach angegriffen werden
  - weil $(p^c)^x = p$ und $\left((p \cdot q)^c\right)^x = (p^c \cdot q^c)^x$
  - Angriff erfolgreich wenn durch q dividiert wird (Known-Plaintext-Insertion-Angriff)
- zum Härten des Systems
  - Zufallszahl zusätzlich zum Klartext verschlüsseln (in den vorherigen Diagrammen bereits enthalten)
  - Hash-Funktion verwenden

    → Verschlüsseln: berechne $\left(n, p, \text{hash}(n, p)\right)^c$ und übertrage es

    → Signieren: berechne $\left(\text{hash}(n, p)\right)^x$ und übertrage es mit p

Übungsmaterial

Unter

https://www.cryptool.org/de/ct2/

steht umfangreiches Übungsmaterial
zum Thema Sicherheit bereit.

Sonstiges

Web Crypto API

- Werkzeug zum
  - Schlüsseltausch
  - clientseitigen Signieren
  - clientseitigen Verschlüsseln
- unterstützt deshalb auch alle wesentlichen Sicherheitsalgorithmen, insbesondere Zufallszahlengenerator (PRNG) und Hash-Funktionen

Beispiel (PBKDF2-Schlüsselerzeugung):

```
(async () => {
const enc = new TextEncoder();
const pw = 'T0p5ecret!';
const key = await crypto.subtle.importKey(
   'raw', enc.encode(pw),
   'PBKDF2', false,
   ['deriveKey']
);
// … anschließend kann der Schlüssel benutzt werden
})();
```

Web Crypto API: Zufallszahlen

```
const randomNumbers = new Uint32Array(1);
crypto.getRandomValues(randomNumbers);
console.log(randomNumbers[0]);
```

Web Crypto API: Signatur

```
(async () =>{
   const message = (new TextEncoder()).encode('Hallo Welt!');
   // Schlüssel generieren
   const key = await crypto.subtle.generateKey(
      {
         name: 'HMAC',
         hash: {name: 'SHA-256'}
      }, false,
      ['sign', 'verify']
   );
   // Nachricht signieren
   const signature = await crypto.subtle.sign(
      {name: 'HMAC'},
      key, message
   );
   // Signatur prüfen
   const isValid = await crypto.subtle.verify(
      {name: 'HMAC'},
      key, signature, message
   );
   console.log('isValid?', isValid);
})();
```

Web Crypto API: Verschlüsselung

```
(async () => {
    const enc = new TextEncoder();
    const dec = new TextDecoder();
    const pw = 'T0p5ecret!';
    // Hauptschlüssel erzeugen
    const key1 = await crypto.subtle.importKey(
        'raw', enc.encode(pw),
        'PBKDF2', false,
        ['deriveKey']
    );
    // Sitzungsschlüssel erzeugen
    const key2 = await crypto.subtle.deriveKey(
        {
            name: "PBKDF2", hash: "SHA-256",
            salt: crypto.getRandomValues(new Uint8Array(10)),
            iterations: 250000
        },
        key1, { name: "AES-GCM", length: 256 },
        false, ['encrypt', 'decrypt']
    );
    const iv = crypto.getRandomValues(new Uint8Array(10));
    const encryptedMessage = await crypto.subtle.encrypt(
        { name: "AES-GCM", iv },
        key2, new TextEncoder().encode('Hallo Welt!')
    );
    console.log('encrypted data: ', new Uint8Array(encryptedMessage));
    const decryptedBuffer = await window.crypto.subtle.decrypt(
        { name: "AES-GCM", iv: iv },
        key2, encryptedMessage
    );
    const decryptedMessage = dec.decode(decryptedBuffer);
    console.log('decrypted data: ', decryptedMessage);
})();
```

security.txt

- vorgeschlagener Standard ähnlich `robots.txt`
- zentrale Datei (`security.txt`) enthält alle wichtigen Informationen
  - URL des PGP/GnuPG-Schlüssels
  - URL der Sicherheitskonfiguration
  - Kontakt für Responsible Disclosure
  - etc.pp.
- Näheres unter https://securitytxt.org

IT-Grundschutz

Mindestmaß für IT-Sicherheit

- definiert in BSI IT-Grundschutz
  - 200-1 (Managementsysteme für Informationssicherheit (MSIS))
  - 200-2 (IT-Grundschutzmethodik)
  - 200-3 (Risikomanagement)
  - 100-4 (Notfallmanagement)
    bzw. im Community-Draft 200-4 (Business Continuity Management)
- Baukastenprinzip

→ https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/it-grundschutz-kompendium_node.html