IT Security

Inhalt

1 Ziele

Ziel dieses Kapitels ist es, die Bedeutung der Informationssicherzeit zu verstehen und durch Basics and Best Practices zu vertiefen.

2 Informationssicherheit

Als *Informationssicherheit* (engl. *Computer Security*) bezeichnet man Eigenschaften von technischen oder nicht-technischen Systemen zur Informationsverarbeitung, -speicherung und -lagerung, die die **Schutzziele** *Vertraulichkeit* (engl. ***C****onfidentiality*), *Integrität* (engl. ***I****ntegrity*) und *Verfügbarkeit* (engl. ***A****vailability*) sicherstellen. Diese sind die wichtigsten drei Prinzipien und werden auch als *CIA-Triade* bezeichnet.

Abbildung 1: CIA-Triade

Im Allgemeinen dient die *Informationssicherheit* dem Schutz vor Gefahren bzw. Bedrohungen, der Vermeidung von wirtschaftlichen Schäden und der Minimierung von Risiken[1].

2.1 Vertraulichkeit

Vertraulichkeit ist definiert als **Schutz vor unbefugter Preisgabe von Informationen** und hat die Aufgabe, diese nur berechtigten Personen zur Verfügung zu stellen bzw. offen zu legen: Nur der Absender und der vorgesehene Empfänger sollten in der Lage sein, den Inhalt einer übertragenen Nachricht zu verstehen. Weil Lauscher die Nachricht abfangen könnten, bedeutet dies notwendigerweise, dass die Nachricht *verschlüsselt* sein muss, sodass eine abgefangene Nachricht nicht von einem Eindringling verstanden

[1] Siehe Informationssicherheit – Wikipedia

werden kann. Dieser Aspekt der *Vertraulichkeit* ist wahrscheinlich die am häufigsten wahrgenommene Bedeutung des Begriffes *Informationssicherheit*.

2.2 Integrität

Integrität wird definiert als **Verhinderung unautorisierter Modifikation von Information** und stellt sicher, dass diese nicht unbemerkt verändert werden können, sondern stets in Originalform verfügbar sein sollen: *Integrität* bedeutet also die Korrektheit bzw. Unversehrtheit von Daten. Diese Nachrichtenintegrität wird technisch durch Erweiterungen der Prüfsummentechnik, Nachrichtenauthentifizierungscode, Quittierungsmeldungen oder Sequenzennummern realisiert.

2.3 Verfügbarkeit

Verfügbarkeit ist die Eigenschaft eines Systems, sämtliche Daten und Funktionen zu einem bestimmten Zeitpunkt zur Verfügung zu stellen. Dies erfordert *Authentifizierung* und *Autorisierung*.

2.4 Weitere Schutzziele

Neben der *CIA-Triade* gibt es *weitere Schutzziele* wie *Authentizität* oder *Verbindlichkeit*[2].

Authentizität

Unter der *Authentizität* (engl. *Authenticity*) eines Objekts bzw. Subjekts wird die Echtheit und Glaubwürdigkeit des Objekts bzw. Subjekts verstanden: Die Identität der berechtigten Personen oder Systeme muss also zweifelsfrei nachgewiesen sein, durch Kenntnis eines Geheimnisses (Passwort), den Besitz eines Gegenstands (Schlüssel) oder körperliche Eigenschaften (Fingerabdruck).

Verbindlichkeit

Verbindlichkeit (engl. Liability) bedeutet, dass es nicht möglich ist, ausgeführte Handlungen wie das Erzeugen, Lesen, Übertragen, Ändern und Löschen abzustreiten.

[2] Quelle: Wirtschaftsinformatik für Dummies

3 Kryptographie

Kryptographie ist die Wissenschaft der *Verschlüsselung* von Informationen[3] und befasst sich mit der Konzeption, Definition und Konstruktion von sicheren Informationssystemen. Im Folgenden liegt der Schwerpunkt auf der Anwendung der Kryptografie zur Sicherstellung von Vertraulichkeit, wobei die Verschlüsselungstechniken untrennbar mit Nachrichtenintegrität, Authentifizierung und Autorisierung verknüpft sind.

3.1 Grundlagen der Kryptographie

Verschlüsselungstechniken ermöglichen einem Sender das Verbergen von Daten, sodass ein Eindringling keine Information aus den abgefangenen Daten herauslesen kann. Der Empfänger muss in der Lage sein, die Originaldaten aus den verschlüsselten Daten wiederherzustellen.

Alle Verschlüsselungsalgorithmen haben gemein, dass dabei *etwas* durch *etwas anderes* ersetzt wird: Beispielsweise wird ein Teil eines **Klartextes** genommen und dann der entsprechende **Chiffretext** berechnet, aus dem die verschlüsselte Nachricht besteht. Ein Vertreter der *klassischen Verschlüsselung* ist die *monoalphabetische Substitution*[4]: Dabei kann jeder Buchstabe jeden anderen Buchstaben ersetzen, solange jeder Buchstabe einen eindeutigen Ersatzbuchstaben besitzt, und umgekehrt. Abbildung 2 zeigt eine mögliche Substitutionsregel, um Klartext zu verschlüsseln.

```
Buchstabe im Klartext:    a b c d e f g h i j k l m n o p q r s t u v w x y z
Buchstabe im Chiffretext: m n b v c x z a s d f g h j k l p o i u y t r e w q
```

Abbildung 2: Eine monoalphabetische Substitution

Die Klartextnachricht `ich liebe Sicherheit` wird nun zu `sba pscbc isbacoacsu`.

Die *modernen Verschlüsselungstechniken* werden in *symmetrische* und *asymmetrische Schlüssel* unterteilt. Bei *symmetrischen Schlüsseln* verwenden die Teilnehmer denselben Schlüssel (Private-Key), der geheim ist. In *asymmetrischen* Systemen (Public-Key Systeme) wird ein Schlüsselpaar eingesetzt: Einer dieser Schlüssel ist öffentlich bekannt (Public-Key). Der andere Schlüssel ist nur dem Besitzer bekannt (Private-Key).

[3] Siehe Kryptographie – Wikipedia

[4] Siehe: Monoalphabetische Substitution – Wikipedia

3.2 Symmetrische Verschlüsselung

Die Verschlüsselungsverfahren der *symmetrischen Kryptographie* (*Private-Key-Verschlüsselung*) arbeiten mit einem **einzigen Schlüssel**, der bei der Ver- und Entschlüsselung vorhanden sein muss. Vor der verschlüsselten Datenübertragung müssen sich die Kommunikationspartner auf den Schlüssel einigen und diesen austauschen. Diese Verfahren sind schnell und bei entsprechend langen Schlüsseln bieten sie auch eine hohe Sicherheit. Vertreter der *symmetrischen Kryptographie* sind *Blockchiffren*[5].

Blockchiffren

Bei *Blockchiffren* werden die zu verschlüsselnde Nachricht in Blöcken von je *k* Bit verarbeitet. Beträgt zum Beispiel *k = 64*, dann wird die Nachricht in Blöcke von 64 Bit Länge aufgeteilt und jeder Block wird unabhängig von den anderen bearbeitet. Beim Verschlüsseln eines Blocks verwendet die Chiffre eine Eins-zu-eins-Zuordnung, um einen *k*-Bit-Block des Klartextes auf einen *k*-Bit-Block des Chiffretexts abzubilden.

Dazu ein Beispiel: Nehmen Sie an, dass *k = 3*, sodass die Blockchiffre 3 Bit lange *Eingaben* als Klartext auf 3 Bit lange *Ausgaben* als Chiffretext abbildet. Eine mögliche Zuordnung enthält Tabelle 1.

Eingabe	Ausgabe	Eingabe	Ausgabe
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

Tabelle 1: 3-Bit-Blockchiffre

Beachten Sie, dass dies eine Eins-zu-eins-Zuordnung ist, das heißt, die *Ausgabe* ist bei jeder Eingabe anders. Diese Blockchiffre unterteilt die Nachricht in 3 Bit lange Blöcke und verschlüsselt jeden Block entsprechend der oben abgebildeten Zuordnung. Sie können nachprüfen, dass die Nachricht `010 110 001 111` zu `101 000 111 001` verschlüsselt wird.

Im Beispiel gibt es $2^3 = 8$ mögliche Eingaben und es können $8! = 40.320$ verschiedene Kombinationen für Schlüssel permutiert werden. Die *Brute-Force-Methode*[6] ist dabei eine beliebte Angriffsmethode, den Chiffretext zu entschlüsseln. Bei 40.320 Kombinationen gelingt dies sehr schnell. Um solche Angriffe zu vereiteln, verwenden Blockchiffren normalerweise viel größere Blöcke, die aus *k = 64* Bit oder mehr bestehen.

[5] Siehe: Blockverschlüsselung – Wikipedia

[6] Siehe: Brute-Force-Methode – Wikipedia

Heute gibt es eine ganze Reihe beliebter *Blockchiffren*, darunter *AES (**A**dvanced **E**ncryption Standard), DES (**D**ata **E**ncryption **S**tandard)* oder *3DES*.

AES

AES ist ein Produktverschlüsselungsverfahren, welches in mehreren Runden die Bits transformiert. Dazu wird zunächst der Klartext in Blöcke mit 128-Bit eingeteilt. Die Schlüssel können 128, 192 und 256 Bit lang sein. Die Anzahl der Transformationsrunden hängt dabei von der Block- und der Schlüssellänge ab und beträgt 10, 12 oder 14. Abbildung 3 zeigt beispielhaft das Verfahren mit 64-Bit-Blöcken:

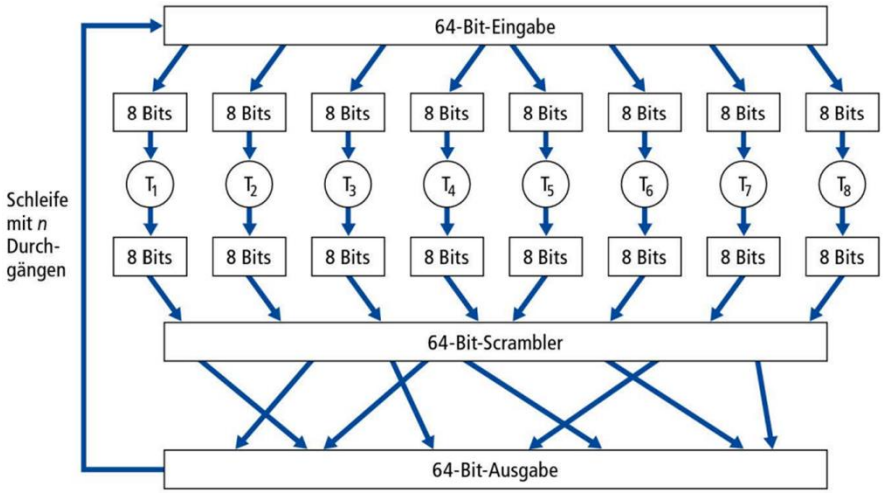

Abbildung 3: Beispiel einer Blockchiffre mit 64-Bit-Blöcken

Der Brute-Force-Angriff auf jede dieser Chiffren besteht darin, alle Schlüssel zu durchlaufen und den Entschlüsselungsalgorithmus mit jedem Schlüssel anzuwenden. Beachten Sie, dass es bei einer Schlüssellänge von *n* genau 2^n mögliche Schlüssel gibt. *NIST*[7] schätzt, dass eine Maschine, die einen *56-Bit-DES* in einer Sekunde knacken könnte (also alle 2^{56} Schlüssel in einer Sekunde testet), etwa 149 Quadrillionen Jahre benötigen würde, um einen *128-Bit-AES-Schlüssel* zu knacken.

3.3 Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselungsverfahren (*Public-Key-Verschlüsselung*) arbeiten mit *Schlüsselpaaren*. Ein Schlüssel ist der öffentliche Schlüssel (*Public-Key*), der andere ist der private Schlüssel (*Private-Key*). Dieses Schlüsselpaar hängt über einen mathematischen Algorithmus eng zusammen. Daten, die mit dem öffentlichen Schlüssel verschlüsselt werden, können nur mit dem privaten Schlüssel entschlüsselt werden. Des-

[7] **N**ational **I**nstitute of **S**tandards and **T**echnology, „Advanced Encryption Standard (AES)", FIPS 197, Advanced Encryption Standard (AES) (nist.gov)

halb muss der private Schlüssel vom Besitzer des Schlüsselpaares geheim gehalten werden.

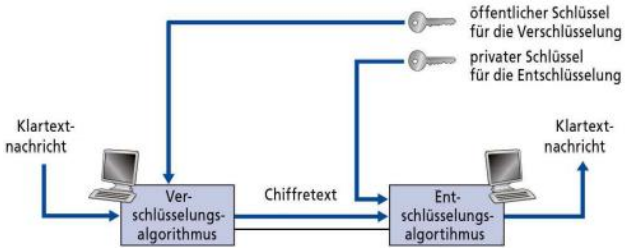

Abbildung 4: Asymmetrische Verschlüsselung

Das bekannteste asymmetrische Verschlüsselungsverfahren ist sicherlich *RSA* (benannt nach *R*ivest-*S*hamir-*A*dleman)[8]. Diese Methode nützt die Tatsache, dass es sehr schwierig ist große Zahlen in ihre Primfaktoren zu zerlegen. So besteht bei diesem Verfahren der öffentliche Schlüssel im Wesentlichen aus dem Produkt zweier großer Primzahlen. Um nun vom *Public-Key* auf den *Private-Key* schließen zu können, müsste eine solche Zerlegung gefunden werden.

RSA

Um den *Public-Key* und den *Private-Key* zu wählen, sind folgende Schritte durchzuführen:

1. Wähle zwei Primzahlen p und q.
2. Berechne $n = p * q$ und $z = (p - 1) * (q - 1)$.
3. Wähle eine Zahl *e*, kleiner als *n*, die keine gemeinsamen Primfaktoren mit *z* hat (ausser *1*). (In diesem Fall bezeichnet man *e* und *z* als relative Primzahlen zueinander). Der Wert wird mit *e* bezeichnet, weil dieser Wert bei der *Verschlüsselung* (*encryption*) verwendet wird.
4. Suche eine Zahl d, sodass $e * d - 1$ ohne Rest durch z teilbar ist. Der Buchstabe d wird benutzt, weil dieser Wert bei der *Entschlüsselung* (***D**ecryption*) verwendet wird. Anders ausgedrückt gesucht wird bei einem gegebenen e ein d, sodass

$$e * d \; mod \; z = 1$$

5. Der öffentliche Schlüssel ist das Zahlenpaar (n, e) und sein privater Schlüssel ist das Zahlenpaar (n, d).

Als einfaches Beispiel sei $p = 5$ und $q = 7$. Dann sind $n = 35$ und $z = 24$. Wähle $e = 5$, da 5 und 24 keine gemeinsamen Faktoren haben. $d = 29$, denn $5 * 29 - 1$ (also $e * d - 1$) ist genau durch 24 teilbar. Folglich werden das Zahlenpaar $(n = 35, e = 5)$ als

[8] Siehe: RSA-Kryptosystem – Wikipedia

Public-Key und *(n = 35, d = 29)* als *Private-Key* eingereiht. Die Tabelle 2 zeigt die Verschlüsselung eines Klartextes und Tabelle 3 die Entschlüsselung des Chiffretextes auf:

m: Klartext	m^e	$c = m^e$ modulo n: Chiffretext
12	248832	17
9	159049	4
5	3125	10
2	32	32

Tabelle 2: RSA-Verschlüsselung mit und $e = 5$

Chiffre-text	c^d	$m = c^d$ modulo n: Klartext
17	481968572106750915091509141182522307169 7	12
4	288230376151711744	9
10	100000000000000000000000000000	5
32	44601490397061246283071436545296723011960832	2

Tabelle 3: RSA-Entschlüsselung mit $n = 35$ und $d = 29$

4 Nachrichtenintegrität und digitale Unterschriften

In diesem Abschnitt wird eine beliebte Technik zur Sicherstellung der Nachrichtenintegrität beschrieben, die beispielsweise von vielen sicheren Netzwerkprotokollen eingesetzt wird. Ferner werden digitale Unterschriften erklärt. Hierfür werden *kryptographische Hash-Funktionen*[9] eingesetzt.

4.1 Kryptographische Hash-Funktionen

Wie Abbildung 5 zeigt, nimmt eine Hash-Funktion eine Eingabe m variabler Länge und berechnet eine Zeichenkette $H(m)$ fester Länge, die als *Hash* bezeichnet wird.

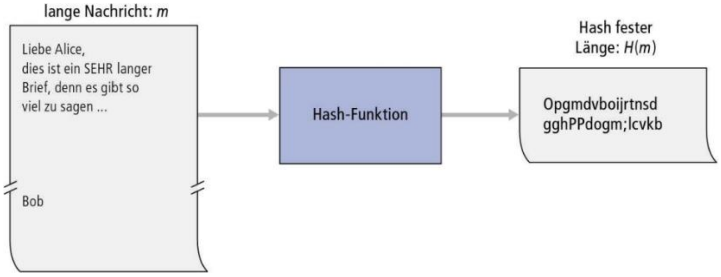

Abbildung 5: Hash-Funktion

Von einer *kryptografischen Hash-Funktion* wird erwartet, dass sie zusätzlich folgendes Merkmal aufweist:

- Es ist mit realistischem Berechnungsaufwand **nicht** möglich, zwei verschiedene Nachrichten x und y zu finden, sodass $H(x) = H(y)$.

Einfach ausgedrückt bedeutet diese Eigenschaft, dass es einem Eindringling unmöglich ist, eine durch die Hash-Funktion geschützte Nachricht durch eine andere zu ersetzen. Das heisst, wenn $(m, H(m))$ die Nachricht und der Hash-Wert der vom Absender erzeugten Nachricht sind, dann kann ein Eindringling keine andere Nachricht y herstellen, die denselben Hash-Wert aufweist wie die Originalnachricht.

Beispiel: Prüfsumme

Es sollen Prüfsummen gebildet, indem jedes Zeichen als ein Byte definiert wird und diese Bytes innerhalb von 4 Byte langen Blöcken aufsummiert werden. Die Zeichenkette

[9] Siehe: Kryptographische Hashfunktion – Wikipedia

`IOUl00.99BOB` in ASCII-Darstellung[10] (in hexadezimaler Schreibweise) dieser Buchstaben lautet `49, 4F, 55, 31, 30, 30, 2E, 39, 39, 42, 4F, 42`.

Nachricht				ASCII-Darstellung				
I	O	U	1	49	4F	55	31	
0	0	.	9	30	30	2E	39	
9	B	O	B	39	42	4F	42	
				B2	C1	D2	AC	**Prüfsumme**

Nachricht				ASCII-Darstellung				
I	O	U	9	49	4F	55	39	
0	0	.	1	30	30	2E	31	
9	B	O	B	39	42	4F	42	
				B2	C1	D2	AC	**Prüfsumme**

Abbildung 6: Die Prüfsumme der ursprünglichen und geänderten Nachricht sind identisch!

Der obere Teil von Abbildung 6 zeigt, dass die 4-Byte-Prüfsumme dieser Nachricht `B2, Cl, D2, AC` lautet. Eine geringfügig geänderte Nachricht wird in der unteren Hälfte von Abbildung 6 gezeigt. Die Nachrichten `IOUl00.99BOB` und `I-OU900.19BOB` haben dieselbe Prüfsumme. Daher verletzt dieser einfache Prüfsummenalgorithmus die Gewährleistung von Nachrichtenintegrität: Sind die Originaldaten bekannt, dann ist es einfach, einen anderen Datensatz mit derselben Prüfsumme zu finden.

Um Sicherheit bieten zu können, brauchen man eindeutig eine leistungsfähigere Hash-Funktion als die Prüfsumme. Vertreter davon sind der *MD5-Hash-Algorithmus* (***M**essage-**D**igest #5 Algorithm*)[11] oder der *SHA* (***S**ecure **H**ash **A**lgorithm*)[12].

MD5 berechnet einen 128-Bit-Hash-Wert in einem vierstufigen Prozess. Die Eingabe erfolgt in 512 Bit Blöcken, wobei kürzere Nachrichten aufgefüllt werden. *SHA-1* erzeugt einen 160 Bit langen Hash-Wert. Seine neueren Verwandten haben längere Hash-Werte (die *SHA-2-Familie*, die z.B. *SHA-256* und *SHA-512* umfasst). Grössere Ausgabelängen bedeuten bei Hash-Algorithmen eine höhere Sicherheit!

[10] Siehe: American Standard Code for Information Interchange – Wikipedia

[11] Siehe: Message-Digest Algorithm 5 – Wikipedia

[12] Siehe: Secure Hash Algorithm – Wikipedia

4.2 Nachrichtenauthentifizierungscode

Ein Nachrichtenauthentifizierungscode (engl. ***M**essage **A**uthentication **C**ode*, kurz *MAC*) ist eine Hash-Funktion *H(.)*, die einen *Authentifizierungsschlüssel s* für die Verifikation des Hashwertes enthält. Dadurch wird Authentizität ohne Geheimhaltung erreicht. Mithilfe eines *MACs* können die übertragenen Nachrichten beglaubigt werden, ohne komplexe Verschlüsselungsalgorithmen einsetzen zu müssen. Einzelne Benutzer können also mit *MACs* überprüfen, ob ihre Nachrichten geändert wurden (z.B. von Malware-Software). Abbildung 7 zeigt schematisch den Ablauf einer Nachrichtenübertragung mit *MACs*.

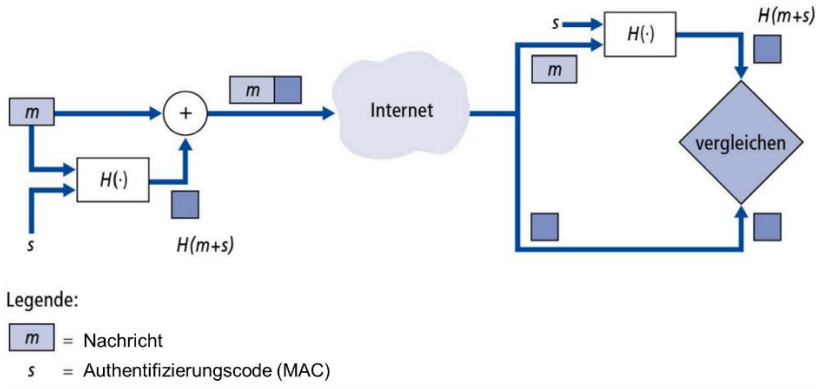

Abbildung 7: Nachrichtenübermittlung mit *MACs*

Der wichtigste Standard ist heute der *Hash-basierte Nachrichtenauthentifizierungscode* (engl. *Hash-based Message Authentication Code*, kurz *HMAC*) [13], der entweder mit *MD5* oder *SHA-1* verwendet werden kann. *HMAC* wendet die Hash-Funktion sogar zweimal auf die Daten und den Authentifizierungsschlüssel an und kann beispielsweise in den Protokollen TLS und SSH eingesetzt werden.

4.3 Digitale Unterschriften

Wie oft haben Sie in der letzten Woche mit ihrem Namen ein Stück Papier unterschrieben, beispielsweise Schecks, Kreditkartenbelege, juristische Dokumente, Briefe, etc.? Mit ihrer Unterschrift stellen Sie sicher, dass Sie (und niemand anderes) den Inhalt des Dokuments bestätigt oder damit einverstanden ist.

Auch in der digitalen Welt benötigt man oft einen Hinweis auf den Erzeuger oder Besitzer eines Dokuments bzw. man möchte sein Einverständnis mit dem Inhalt eines Dokumentes deutlich machen. Eine *digitale Unterschrift* (engl. ***D**igital **S**ignature*, kurz *DS*) ist die kryptografische Technik, um dieses Ziel in der digitalen Welt zu erreichen. Ge-

[13] Siehe: HMAC – Wikipedia

nau wie bei handschriftlichen Signaturen sollten digitale Unterschriften *verifizierbar* und *fälschungssicher* sein. Die Signierung erfolgt mit dem RSA-Verfahren.

Ein Problem bei der Signierung von Daten mittels Verschlüsselung besteht darin, dass Ver- und Entschlüsselung zu rechenintensiv ist. Daher berechnet ein Hash-Algorithmus für eine Nachricht von beliebiger Länge einen „*Fingerabdruck*“ fester Länge. Der „*Fingerabdruck*“ wird mit dem *Private-Key* verschlüsselt und gilt als *digitale Unterschrift*. Der Empfänger des Dokumentes entschlüsselt die *digitale Unterschrift* mit dem *Public-Key* und überprüft, ob der Hash-Wert, den er selbst erstellt, mit dem entschlüsselten Wert übereinstimmt. Bei Übereinstimmung der Hash-Werte, ist die Integrität des Erzeugers der Nachricht gewährleistet (siehe Abbildung 8).

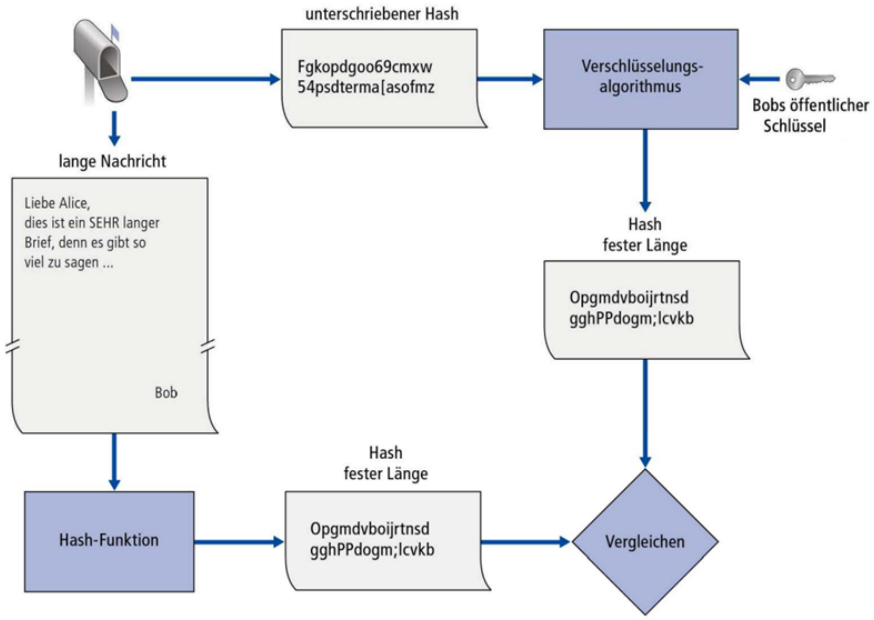

Abbildung 8: Überprüfung einer digital unterschriebenen Nachricht

Für eine *Public-Key*-Verschlüsselung ist es allerdings unbedingt erforderlich, den Nachweis der Authentizität durch digitale Zertifikate zu leisten.

4.4 Digitale Zertifikate

Zertifikate sind so etwas ähnliches wie Ausweise. Die wesentlichen Inhalte eines solchen Zertifikates sind der Zertifikatsinhaber, deren Authentizität durch diesen Ausweis bestätigt wird, deren *Public-Key*, die Zertifizierungsstelle und deren digitale Unterschrift über das Zertifikat. Der Aufbau von Zertifikaten ist standardisiert (siehe Abbildung 9)[14].

[14] Siehe: X.509 – Wikipedia

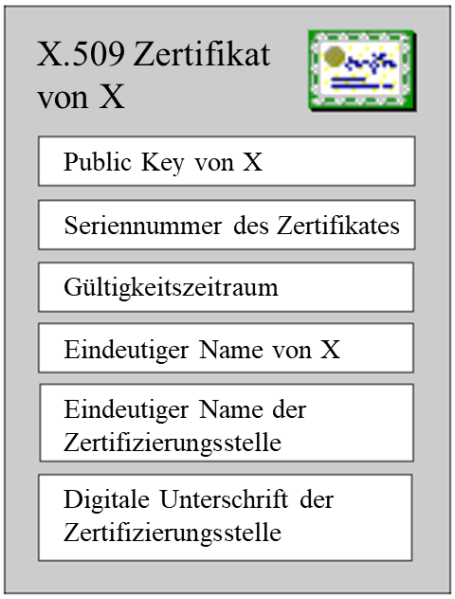

Abbildung 9: Wesentliche Inhalte eins *X.509* Zertifikates,
wobei *X* der Zertifikationsinhaber ist

Die Aufgabe, die ein Zertifikat erledigt, ist die eindeutige Zuordnung eines *öffentlichen Schlüssels* zu einer bestimmten Institution X (siehe Abbildung 9). Dass diese Zuordnung korrekt ist und dass die Institution, die den passenden *privaten Schlüssel* besitzt, tatsächlich existiert, dafür verbürgt sich die ausstellende *Zertifizierungsstelle (*engl. ***C****ertification* ***A****uthority*, kurz *CA)*. Diese fügt dem ausgestellten Zertifikat seine digitale Unterschrift hinzu, wodurch es für Dritte ohne die Kenntnis des *privaten Schlüssels* der Zertifizierungsstelle unmöglich wird, das Zertifikat zu verändern. *CAs* sind allgemeine Anbieter für Vertrauensdienste, Berufsverbände (z.B. Wirtschaftsprüfer, Notare usw.), Personal und IT-Abteilungen von Unternehmen, Behörden.

5 Authentifizierung und Autorisierung

Durch den Prozess der *Authentifizierung* wird die Identität innerhalb eines IT-Systems überprüft (siehe Abbildung 10):

- Aus Sicht des **Benutzers** ist die *Authentisierung* die Prüfung des Identitätsnachweises auf Authentizität.
- Aus Sicht des **IT-Systems** findet zum einem die *Authentifizierung* und zum anderen die *Autorisierung* statt.
    a. Bei der *Authentifizierung* handelt es sich um die Überprüfung der tatsächlichen Identität des Benutzers. Sowohl Sender als auch Empfänger sollten in der Lage sein, die *Identität* des jeweils anderen Kommunikationspartners zu bestätigen.
    b. Die *Autorisierung* wird als eine Sicherheitsfunktion betrachtet. Es geht darum, welcher Benutzer welchen Zugriff auf welche Funktionen hat. Dies erfolgt durch Zuweisen und wiederholtes Überprüfen von Zugriffsrechten.

Abbildung 10 stellt schematisch den Prozessablauf der Authentisierung dar.

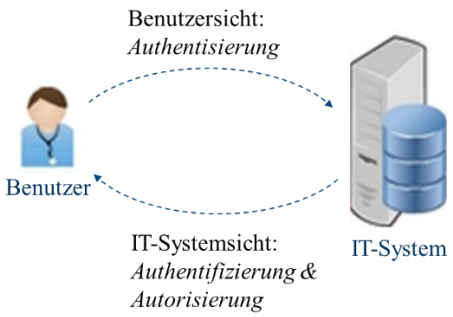

Abbildung 10: Prozessablauf der Authentifizierung

Im Authentifizierungsverfahren wird zwischen *Wissen*, *Besitz* und *Sein* unterschieden, wie der Nachweis der Echtheit der digitalen Identität zu überprüfen ist.

- *Wissen (what you know):* Hier wird der Nachweis durch die Kenntnis einer Information auf die Echtheit eines Nutzers geprüft. Beispiele sind Passwort, PIN, Antwort auf eine bestimmte Frage (Sicherheitsfrage) usw.
- *Besitz (what you have):* Die Verwendung eines Besitztums gilt als Nachweis für das Authentifizierungsverfahren. Beispiele sind Personalausweis, SIM-Karte im Smartphone, Hardware-Sicherheitsmodule (Smartcard, USB-Stick) usw.

- *Sein (what you are):* Bei diesem Authentifizierungsverfahren muss der Nutzer als Nachweis gegenwärtig sein. Beispiele sind Fingerabdruck, Retina Scan usw.
- *Weitere Verfahren:* Es können noch weitere unterstützende Faktoren für die Beurteilung der Echtheit des Nutzers herangezogen werden: Beispiele sind vergangene Transaktionen des Nutzers (Reputation), verwendete Endgeräte und Software des Nutzers (Technologie), Standort und Zeit des Authentisierungsprozesses.

Im Autorisierungsverfahren ist die *rollenbasierte Zugriffskontrolle* (engl. ***R**ole **B**ased **A**ccess **C**ontrol*, kurz *RBAC*)[15] eine Möglichkeit, die Zugriffsberechtigung der Nutzer innerhalb eines IT-Systems zu erteilen. Dieses *Sicherheits- und Berechtigungskonzept* ermöglicht die Vergabe von Rollen und Berechtigungen. Dabei werden die Zugriffsrechte anhand eines definierten Rollenmodells vergeben.

Zweifaktor Authentifizierung (2FA)

Für die *2FA* werden zwei unterschiedliche und unabhängige Faktoren für den Nachweis der Echtheit der digitalen Identität verwendet. Eine häufige Variante die mit Besitz und Wissen: Beispielsweise kann eine Smartcard (Hardware-Sicherheitsmodul) durch einen PIN aktiviert werden. Dabei muss der Nutzer die PIN kennen (Wissen) andererseits das Hardware-Sicherheitsmodul haben (Besitz).

[15] Siehe: Role Based Access Control – Wikipedia

6 Schlüsselverwaltungssysteme

Ein *Schlüsselverwaltungssystem* (engl. ***K**ey **M**anagement **S**ystem*, kurz *KMS*) verwaltet die für kryptographische Verfahren benötigten Schlüssel und Zertifikate in einem Unternehmen an zentraler Stelle, erinnert rechtzeitig vor deren Ablauf und unterstützt den kompletten *Key Lifecycle*. Einerseits soll der Überblick über die im Unternehmen im Einsatz befindlichen Schlüssel und Zertifikate gewährleistet, anderseits deren Gültigkeit überwacht und kontrolliert werden. Das *KMS* stellt sicher, dass Schlüssel echt sind und geheim gehalten werden. Es kann eine große Anzahl von Schlüsseln generieren, aufbewahren, bereitstellen, austauschen und schützen.

6.1 Schlüsselmanagement-Protokoll KMIP

Anwendungen und Systeme müssen Informationen mit dem *KMS* auf sicherem Weg austauschen können. Dafür wurde das ***K**ey **M**anagement **I**nteroperability **P**rotocol* (kurz *KMIP*)[16] entwickelt. Über dieses standardisierte Kommunikationsprotokoll können verschiedene Systeme für kryptografische Operationen derart verknüpft werden, damit eine zentrale Schlüsselverwaltung ermöglicht wird.

6.2 Public-Key-Infrastruktur (PKI)

Eine *PKI* dient zum Erstellen und Verwalten von digitalen Zertifikaten über deren gesamten Lebenszyklus, von der Erstellung über die Aufbewahrung und Verwendung bis hin zur Löschung. Dabei kommt es, außer auf die sichere Erstellung und Speicherung gültiger Schlüssel, auch auf die Verifizierung der ursprünglichen Identität der *PKI-Nutzer* an.

Eine *PKI* besteht aus Hardware, Software und einem abgestimmten Regelwerk, der Leitlinie. Diese definiert, nach welchen Sicherheitsregeln die Services für digitale Zertifikate erbracht werden. Dazu zählen das Betriebskonzept der PKI, die Nutzerrichtlinien sowie Organisations- und Arbeitsanweisungen.

[16] Siehe: Key Management Interoperability Protocol - Wikipedia

7 Operative Sicherheit

Wird in einem Computernetzwerk der Datenverkehr, der in das Netz eintritt oder es verlässt, auf seine Sicherheit überprüft, protokolliert, verworfen oder weitergeleitet, dann sind *Firewalls*, *Intrusion-Detection-Systeme (IDS)* und *Intrusion-Prevention-Systeme (IPS)* daran beteiligt.

7.1 Firewall

Eine *Firewall*[17] ist ein Sicherungssystem, das das *Intranet* einer Organisation vom *Internet* abschottet, wobei sie einige Datenpakete passieren lässt und andere blockiert. Eine *Firewall* ermöglicht einem Netzwerkadministrator die Kontrolle des Zugriffs der Außenwelt auf die Ressourcen innerhalb des von ihm verwalteten Netzes, indem das Sicherungssystem die Datenverkehrsströme zu und von diesen Ressourcen regelt (siehe Abbildung 11)

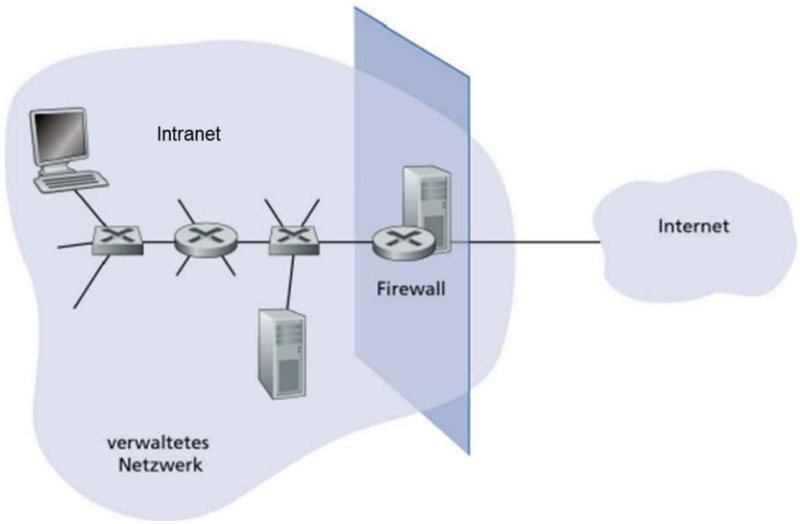

Abbildung 11: *Firewall*, platziert zwischen dem durch einen Administrator verwalteten Netzwerk und der Außenwelt

Eine *Firewall* hat drei zentrale Ziele:

1. Der komplette Datenverkehr von außen nach innen und umgekehrt passiert die Firewall.
2. Nur berechtigter Datenverkehr, definiert durch die lokale Sicherheitsrichtlinie, darf passieren.

[17] Siehe: Firewall – Wikipedia

3. Die Firewall selbst kann nicht kompromittiert werden, d.h. sie muss installiert und richtig aufgebaut sein (sonst entsteht ein falsches Gefühl der Sicherheit).

Die Aufgaben von Firewall-Systemen sind die *Zugangskontrolle* auf der *Netzwerk-, Nutzer- und Datenebene*.

Auf *Netzwerkebene* wird überprüft, welche IT-Systeme über das Firewall-System miteinander kommunizieren dürfen.

Auf *Nutzerebene* überprüft das Firewall-System, welche Nutzer und IT-System über das Firewall-System eine Kommunikation aufbauen dürfen. Dabei werden die Authentizität des Nutzers sowie der IT-Systeme überprüft.

Auf *Datenebene* überprüft das Firewall-System, welche Daten eines definierten Nutzers und IT-Systems über das Firewall-System übertragen werden dürfen.

Weitere Aufgaben von Firewall-Systemen sind

- die Alarmierung von sicherheitsrelevanten Ereignissen,
- die Beweissicherung und Protokollauswertung von Verbindungsdaten sowie sicherheitsrelevanten Ereignissen,
- die Kontrolle auf der Anwendungsebene (z.B. Spam, Malware),
- die Rechteverwaltung, die festlegt, mit welchen Protokollen, Diensten und zu welchen Zeiten eine Kommunikation erlaubt ist,
- die Entkopplung von Diensten, damit Schwachstellen und Konzeptionsfehler keine Angriffe von aussen ermöglichen.

7.2 Intrusion-Detection-Systeme

Das Firewall-System selbst bietet nur einen sehr geringen Schutz vor internen Angriffen. Um internen Angriffen entgegenzuwirken, müssen weitere, ergänzende Sicherheitsmechanismen eingeführt werden. Hierfür gibt es ein Gerät, das Warnsignale auslöst, wenn es potenziell gefährlichen Datenverkehr beobachtet. Dieses Gerät wird als *Intrusion-Detection-System* (kurz *IDS*, System zur Erkennung von Eindringlingen) bezeichnet und schützt also vor internen und externen Angriffen. Ein Gerät, das dagegen verdächtigen Verkehr herausfiltert, wird als *Intrusion-Prevention-System* (kurz *IPS*, System zur Vorbeugung gegen Eindringlinge) bezeichnet.

Ein *IDS* lässt sich einsetzen, um eine große Zahl von Angriffen zu erkennen, beispielsweise das Ausspähen von Netzwerken (Mapping, etwa mithilfe von nmap[18]), Port Scans,

[18] Siehe: Nmap: the Network Mapper - Free Security Scanner

Scans von TCP-Stacks, DoS-Angriffe[19], Würmer und Viren, Angriffe auf Schwachstellen von Betriebssystemen und Anwendungen.

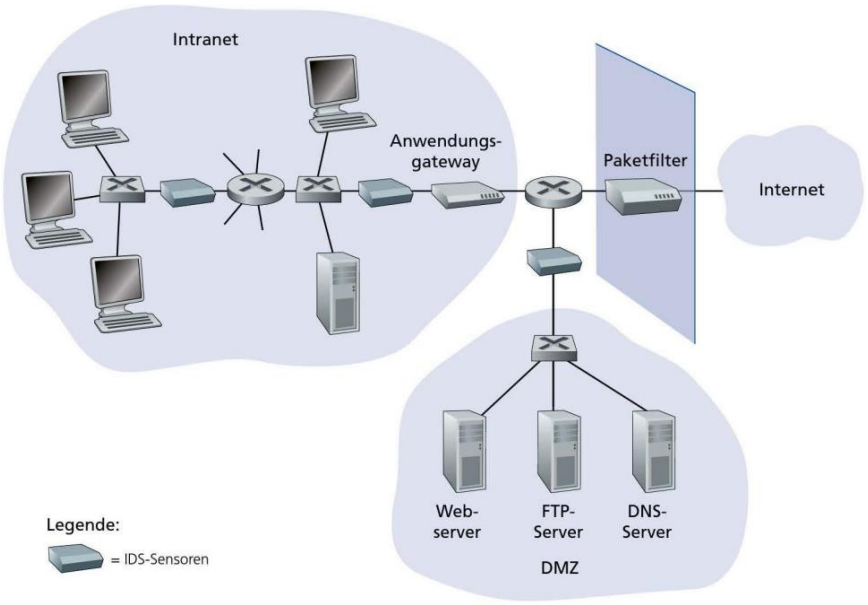

Abbildung 12: Eine Organisation, die einen Filter, ein Anwendungsgateway und IDS-Sensoren einsetzt

[19] Siehe: Denial of Service – Wikipedia

8 Aufgaben

8.1 Aufgabe 1a: Prüfsummenberechnung von Nachrichten

In dieser Aufgabe berechnen Sie zur Überprüfung der Integrität die Prüfsumme (engl. Checksum) einer zusammengesetzten Nachricht (engl. Message). Abbildung 13 stellt schematisch den Ablauf der Prüfsummenbildung dar:

Abbildung 13: Prüfsummenbildung

Message beschreibt die Nachricht, für die eine Prüfsumme berechnet werden soll. *MessageDigest* stellt die kryptographische Hashfunktion SHA-256 bereit. *Digest/Hash* wird als Prüfsumme in einem *Byte-Array* gespeichert.

1. Öffnen Sie mit *BlueJ* aus dem Projektordner …*Security* die Projektvorlage *CheckSum*.
2. Kompilieren Sie die Klasse `CheckSum`.
3. Starten Sie in *BlueJ* `CheckSum` mit der Methode `checkSumOfMessage`. Öffnen Sie anschliessend die Konsole und überprüfen Sie das Ergebnis: Welche Nachricht wurde von der Methode gesendet? Aus wieviel Bytes besteht die Prüfsumme?
4. Studieren Sie nun den Code in der Methode `checkSumOfMessage`.
5. Anschliessend implementieren Sie die Methode `checkSumOfCompositeMessage`: Teilen Sie den Inhalt der Nachricht `message` in drei gleiche Teile `m1`, `m2` und `m3` auf und fügen Sie die Teilnachrichten der Instanz von `MessageDigest` mit der Methode `update(Byte[])` hinzu. Bestimmen Sie die Prüfsumme und geben Sie diese sowie die Länge der Prüfsumme in Bytes und Bits in der Konsole aus.
6. Vergleichen Sie das Ergebnis der Prüfsummen von `checkSumOfMessage` und `checkSumOfCompositeMessage`. Was fällt Ihnen auf?

8.2 Aufgabe 1b: Prüfsummenberechnung von Dateien

Dateien können während eines Downloads abgefangen, manipuliert der zufällig verändert werden. Damit nachprüfbar ist, ob eine Datei unverändert heruntergeladen wurde und die Integrität gewährleistet bleibt, erzeugt der Ersteller der Downloaddatei vor der Übertragung ins Internet mittels eines bestimmten Programmes eine Prüfsumme. Die Prüfsumme, für die zum Download angebotene Datei, stellt er dann zusätzlich zur Downloaddatei im Internet zur Verfügung.

1. Studieren Sie nun in der Projektvorlage *CheckSum* den Code in der Methode `checkSumOfFile`. Welche kryptographische Hashfunktion wird für die Bestimmung der Prüfsumme von der Datei verwendet?
2. Starten Sie in *BlueJ* `checkSumOfFile` und öffnen Sie anschliessend die Konsole mit der erstellten Prüfsumme der Datei.
3. Öffnen Sie die Online-Berechnung und bilden Sie Prüfsumme von der zu überprüfenden Datei *...\Security\Checksum\20230223_171546.jpg*. Vergleichen Sie die Werte und überprüfen Sie, ob diese identisch sind.
4. Ersetzen Sie den Prüfsummen-Algorithmus SHA-256 durch SHA-512 und/oder MD5 vergleichen Sie die Längen der Prüfsummen. Was fällt hier auf?

   Hinweis: Die Bestimmung der Prüfsumme kann ebenfalls durch das Tool GtkHash bestimmt werden.

8.3 Aufgabe 2: Symmetrische Verschlüsselung

Führen Sie eine Verschlüsselung eines Klartextes (engl. Plaintext) mit anschliessender Entschlüsselung des Chiffriertextes (engl. Chiphertext) mit einem AES-Schlüssel durch.

Abbildung 14 stellt schematisch den Ablauf der Symmetrischen Verschlüsselung dar:

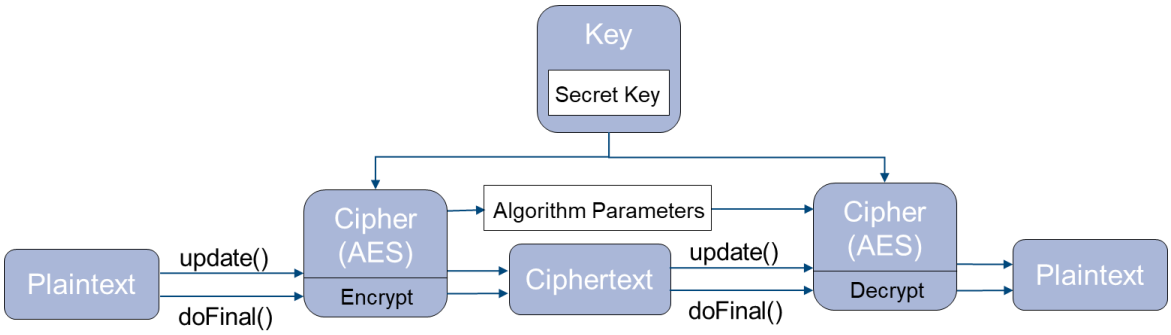

Abbildung 14: Symmetrische Verschlüsselung

Die Klasse *Cipher* stellt den Ver- und Entschlüsselungs-Algorithmus (engl. *Encrypt* and *Decrypt*) zur Verfügung. Dabei ist zu beachten, dass der **gleiche Schlüssel** sowohl für *Encrypt* als auch für *Decrypt* zu verwenden ist:

1. Öffnen Sie dazu mit *BlueJ* aus dem Projektordner …*Security* die Projektvorlage *AES*.
2. Kompilieren Sie die Klasse `AES`.
3. Studieren Sie nun den Code der statischen Methoden `encrypt` und `decrypt`. Aus wieviel Bits besteht der AES-Schlüssel? Was bewirkt `init` von der Instanz-Variable `cipher` in den beiden Methoden `encrypt` und `decrypt`?
4. Definieren Sie einen eigenen Klartext (`plainText`) und einen Schlüssel mit genau 16 Character und weisen Sie den Wert der Variable `myKey` zu. Starten Sie in *BlueJ* `AES` mit der Methode `main`. Öffnen Sie anschliessend die Konsole und überprüfen Sie das Verschlüsselungsergebnis.

8.4 Aufgabe 3: Asymmetrische Verschlüsselung

Führen Sie eine Verschlüsselung eines Klartextes mit anschliessender Entschlüsselung des Chiffriertextes mit einem RSA-Schlüssel durch. Dabei ist zu beachten, dass für die Verschlüsselung ein öffentlicher (engl. *Private Key*) und für die Entschlüsselung ein privater (engl. *Private Key*) Schüssel in Form eines **Schlüsselpaares** zu generieren ist:

1. Öffnen Sie dazu mit *BlueJ* aus dem Projektordner …*Security* die Projektvorlage *RSA*.
2. Kompilieren Sie die Klasse `RSA`.
3. Studieren Sie nun den Code der statischen Methoden `generateKey`, `encrypt` und `decrypt`. Aus wieviel Bits besteht der RSA-Schlüssel? Was bewirkt `init` von der Instanz-Variable `cipher` in den beiden Methoden `encrypt` und `decrypt`?
4. Definieren Sie einen eigenen Klartext. Starten Sie in *BlueJ* `RSA` mit der Methode `main`. Öffnen Sie anschliessend die Konsole und überprüfen Sie das Verschlüsselungsergebnis.

8.5 Aufgabe 4: Verschlüsselte Internet-Kommunikation

Um die Vertraulichkeit und Integrität in der Kommunikation zwischen Webserver und Webbrowser zu ermöglichen, werden die Daten verschlüsselt mit dem HTTPS-Protokoll übertragen. Dabei geschieht die Verschlüsselung mittels SSL/TLS. Eine HTTPS-Verbindung wird durch eine https-URL angewählt und durch das SSL-Logo angezeigt: Dies wird als kleines Schloss-Symbol in der Adresszeile dargestellt.

Selbstsigniertes SSL/TLS-Zertifikat erstellen

Bevor Sie einen HTTPS-Server implementieren und prüfen, ist ein selbstsigniertes SSL/TLS-Zertifikat zu erstellen. Dies bedeutet, dass das Zertifikat seine eigene CA ist und nicht von einer Zertifizierungsstelle ausgestellt wird.

1. Öffnen Sie die Eingabeaufforderung als **Administrator**. Um den aktuellen Java-Bin-Ordner zu ermitteln, geben Sie den Befehl `where java` ein.
2. Wechseln Sie in der Eingabeaufforderung in den Java-Bin-Ordner (z.B. `C:\Program Files\Java\openjdk\bin>` oder `C:\Program Files\Java\jdk-15\bin>`)
3. Für die Zertifikatserstellung geben Sie folgenden Befehl ein:

   ```
   keytool -genkeypair -keyalg RSA -alias selfsigned -keystore testkey.jks -storepass password -validity 360 -keysize 2048
   ```

   Der Befehl fragt Sie nach Ihrem *Vor-* und *Nachnamen*, der *Organisationseinheit*, der *Organisation*, der *Stadt*, dem *Kanton* und dem zweistelligen *Ländercode*. Geben Sie alle Informationen ein und bestätigen Sie ***Ja***[20].

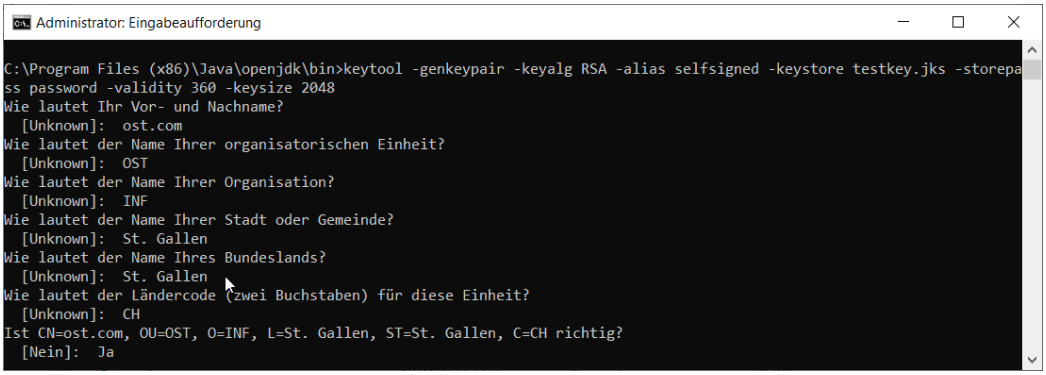

Abbildung 15: Zertifikatserstellung

[20] **Hinweis:** Bei englischer Systemsprache ist mit **Yes** zu bestätigen.

4. Durch *keytool* wird die Datei *testkey.jks* im Java-Bin-Ordner erstellt. Kopieren Sie diese in Ihren Java-Projektordner …*Security**HTTPSServer*, um das Zertifikat für die HTTPS-Verbindung zu verwenden.

HTTPS-Verbindung implementieren prüfen

1. Öffnen Sie dazu mit *BlueJ* aus dem Projektordner …*Security* die Projektvorlage *HTTPSServer*.
2. Kompilieren Sie die Klasse `HTTPSServer`.
3. Studieren Sie nun den Code von der Methode `main`: Für den gesicherten Datenaustausch wird ein Schlüsselspeicher (Klasse `KeyStore`) verwendet. Die Instanz `keyStore` benötigt für das Laden der Zertifikatsdatei *testkey.jks* das Passwort. Weisen Sie das Passwort der Variable `password` zu.

   Hinweis: Studieren Sie dazu den Parameter `-storepass`, welcher Teil der Befehlseingabe für die Zertifikatserstellung ist.
4. Starten Sie in *BlueJ* `HTTPSServer` mit der Methode `main`.
5. Öffnen Sie den Webbrowser und testen Sie die Verbindung mit der Adresse `https://localhost:8080/page`

   Hinweis: Da ein selbstsigniertes Zertifikat eingesetzt wird, erscheint nach der Adresseingabe im Webbrowser die Nachricht «Nicht sicher» – bei einem öffentlichen Zertifikat wird Schloss-Symbol angezeigt.
6. Klicken Sie auf das Symbol und suchen Sie in der Anzeige das Zertifikatssymbol. Öffnen den Zertifikats-Viewer mit Klick auf das Symbol und wählen Sie in der Rubrik *Details* das Zertifikatsfeld *Zertifikatsignaturalgorithmus* aus. Welcher Algorithmus wird hier verwendet?
7. Vervollständigen Sie nun die Klasse `MySecondTestHandler`: Fügen Sie der Instanz `buffer` vom Typ `StringBuffer` folgende HTML-Zeilen für die Servernachricht hinzu:

```
buffer.append("<HTML>\n");
buffer.append("<HEAD><TITLE>HTTPS
   Server</TITLE></HEAD>\n");
buffer.append("<BODY>\n");
buffer.append("<H1>Success!</H1>\n");
```


8. Starten Sie in *BlueJ* `HTTPSServer` mit der Methode `main` und öffnen Sie den Webbrowser. Testen Sie die Verbindung mit der Adresse https://localhost:8080/secondpageund überprüfen Sie die Nachricht.

8.6 Aufgabe 5: Zweifaktor Authentifizierung (2FA)

2FA mit Google Authenticator basiert auf der Generierung von zeitbasierten Einmal-passwörtern (engl. Time-based One-time Passwords, TOTP). Um beispielsweise Zugriff auf die Temperaturwerte eines Sensors zu erhalten, wird TOTP in Kombination mit dem Login für die Authentifizierung benutzt: Dabei wird das zusätzliche Einmal-Passwort nach dem Login für die erweiterte Authentifizierung zur Benützung der Anwendung eingesetzt. Das Einmal-Passwort ist für eine kurze Zeitspanne gültig ist, in der Regel 30 Sekunden.

Installation Google Authenticator-App

Installieren Sie für die Generierung von TOTP die Google Authenticator-App auf Ihrem Smartphone, die im App Store oder Play Market erhältlich ist.

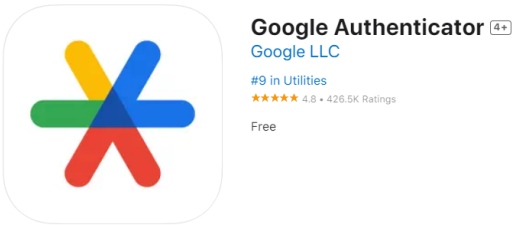

Abbildung 16: Google Authenticator-App

Aktivierung der 2FA

1. Öffnen Sie mit *BlueJ* aus dem Projektordner …*Security* die Projektvorlage *TwoFactorAuthentification*, und kompilieren Sie anschliessend alle Klassen.
2. Starten Sie in *BlueJ* `TwoFactorApp` mit Run JavaFX Application.

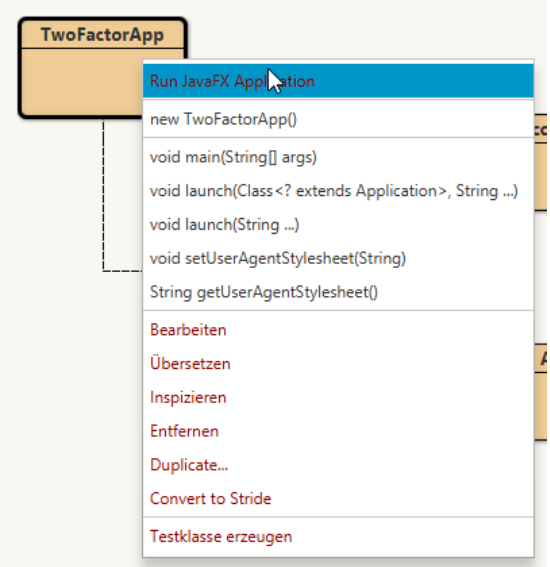

Abbildung 17: Aufruf der 2FA-Applikation

3. Richten Sie unter Rubrik "*Step 1: Sign Up"* Ihr Benutzerkonto ein und bestätigen Sie die Angaben mit *Sign Up.* Sind alle Angaben plausibel, so erscheint ein 32-stelligen Einrichtungsschlüssel als QR-Code und im Textformat.

Abbildung 18: Aufruf der 2FA-Applikation

4. Öffnen Sie nun Ihre Google Authenticator-App: Drücken Sie auf das *Plus-Symbol*, um einen neuen Eintrag hinzuzufügen. Wählen Sie *"Scan a QR code"* oder *"Enter a setup key"*. Sie sollten Ihren Eintrag in der Liste mit einem 6-stelligen Code sehen, der sich alle 30 Sekunden ändert (siehe Abbildung 20)

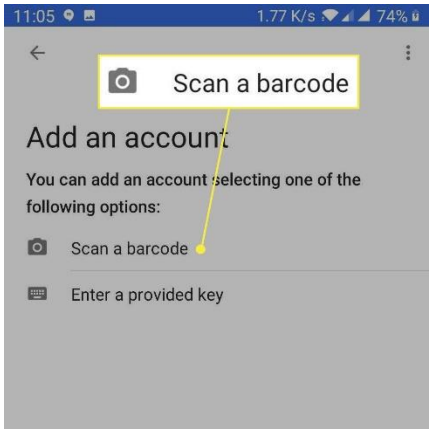

Abbildung 19: Google Authenticator-App – Eintrag Hinzufügen

5. Bestätigen Sie nun die durchgeführte Eintragung in der 2FA-Applikation mit *Confirm* (siehe Abbildung 18), um die Registrierung Ihres Benutzerkontos abzuschliessen.

Login mit 2FA

1. Melden Sie sich nun unter Rubrik *"Step 2: Log In"* mit Mailadresse und Passwort an. Anschliessend tippen Sie unter *"Step 3: Code"* den 6-stelligen TOTP-Code ein und schliessen mit *Confirm* ab.

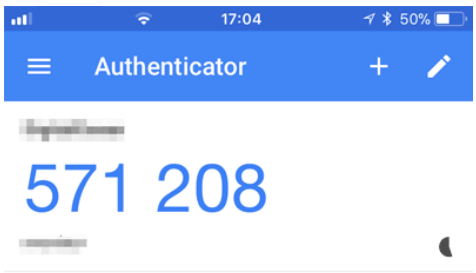

Abbildung 20: Google Authenticator-App – Code

Hinweis: Nach Ablauf der Zeitspanne von 30 Sekunden wird ein neuer TOTP-Code generiert. Achten Sie auch darauf, dass die Uhrzeit Ihres Smartphones sowie Ihres Desktop-Arbeitsplatzes synchronisiert ist.

2. Sie haben eine erfolgreiche 2FA-Authentifizierung durchgeführt und können nun auf die Temperaturwerte zugreifen.

9 Best Practices

Für das Design von sicheren IT- bzw. IoT-Systeme gibt keine Rezepte, dafür aber eine *Best Practice*. Der Begriff *Best Practice*[21] (dt. beste Methode, beste Praxis, beste Vorgehensweise) stammt aus der angloamerikanischen Betriebswirtschaftslehre und bezeichnet bewährte, optimale bzw. vorbildliche Methoden, Praktiken oder Vorgehensweisen im Unternehmen.

Die folgende Auflistung der *Best Practices für Informationssicherheit* sind Empfehlungen von IEEE[22], die von Fachleuten als gut oder notwendig akzeptiert werden:

I. Die Hardware/Software manipulationssicher machen (engl. tamper resistant)

- Viele IoT[23] Geräte sind dauernd (7x24 h) und unbeaufsichtigt im Betrieb
- Es gilt potenzielle Eindringlinge von den Daten fernzuhalten
- **Physische Sicherheit** durch spezielle Gehäuse, USB- und Ethernet-Port Schlösser, abgeschlossene Räume, etc.
- Gerätedeaktivierung, wenn Manipulation detektiert wird
- Starke *boot-level* Passwörter
- Bekannte Schwachstellen absichern (**engl. Hardening**)
  - ✓ Offene TCP/UDP-Ports
  - ✓ Offene serielle Schnittstellen
  - ✓ Password Prompts
  - ✓ Code Injektionen bei Web Servern
  - ✓ unverschlüsselte Kommunikation
  - ✓ etc.
- **Generell:** geschichteter Ansatz anwenden, d.h. eine Vielzahl von Hindernissen dem Angreifer in den Weg stellen, anstelle eines einzelnen Hindernisses

[21] Siehe: Best Practice – Wikipedia

[22] Institute of Electrical and Electronics Engineers: IEEE - The world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

[23] Siehe: IoT-Plattform – Wikipedia

II. Firmware-Updates bereitstellen (conditio-sine-qua-non)

- Schachstellen werden unweigerlich zu Tage treten, wenn das Gerät in Betrieb ist
- Hersteller verdienen kein Geld mit der Bereitstellung von Firmware-Updates, nur mit dem Verkauf von Geräten
- **Geräte brauchen zwingend eine Update-Funktionalität**
- Firmware-Updates nur mit digitaler Unterschrift (man muss die Quelle des Updates überprüfen können)
- Für viele Geräte ist ein ***O**ver-**t**he-**A**ir-Update* (kurz *OTA*)[24] gar nicht vorgesehen (z.B.: LoRa[25])
- **Geräte ohne Update-Mechanismus können gar nicht sicher sein!**
- **Beispiel:** erstmals DDoS[26] Attacke mit IoT Geräten am 21.10.2016 (Mirai botnet)
- Hersteller müssen verpflichtet werden ihre Geräte über einen vereinbarten Lebenszyklus zu überwachen und zu warten
- Der Lebenszyklus eines Gerätes beginnt bereits bei der Herstellung und endet bei der Entsorgung

III. Durchführung von dynamischen Tests sind notwendig

- **Statisches Testen:**
  - ✓ Software-Techniken, bei denen das Testen durchgeführt wird, ohne den Code auszuführen
  - ✓ **Beispiel**: Statische Code Analysis Tools (z.B.: FindBugs), Code Reviews
- Mit statischen Tests können keine Schwachstellen im Gesamtsystem (HW + SW) gefunden werden
- Dynamisches Testen, das den Code auch ausführt, ist notwendig, um Schwachstellen im Code und in der zugrunde liegenden Hardware zu entdecken
- IoT Geräte müssen gründlich getestet werden, um eine Mindestanforderung an Sicherheit zu erfüllen

[24] Siehe: Over-the-Air-Update – Wikipedia
[25] Siehe: Long Range Wide Area Network – Wikipedia
[26] Siehe: Denial of Service – Wikipedia

IV. Datenschutz bei der Entsorgung von Geräten

- IoT Geräte dürfen bei der Entsorgung keinerlei private Daten preisgeben
- Aus nicht fachgerecht entsorgten IoT Geräte können Informationen über den ehemaligen Besitzer und dessen IoT Umgebung extrahiert werden (gilt auch für Geräte, die weitergegeben werden)
- Benutzer verfügen oft über zu wenig Wissen, um zu verstehen welche persönlichen Informationen in den IoT Geräten gespeichert sind
  - ✓ **Beispiel**: Kopierer mit eingebauter Festplatte
- Hersteller sollten Benutzern eine konkrete Beschreibung für die fachgerechte Entsorgung zur Verfügung stellen
- Hersteller sollten die Entsorgung der Geräte anbieten

V. Starke Authentifizierung verwenden

- Authentifizierung: what you know, what you have, what you are, combinations
- IoT Geräte sollten keine einfach zu erratenden Anmeldeinformationen (engl. *Credentials*) verwenden (z.B.: admin/admin)
- IoT Geräte sollten keine Hintertüren (*back doors*) oder versteckte Entwickler Credentials enthalten
- Jedes IoT Geräte sollte eine eindeutige Standard-Benutzername/Passwort Kombination anbieten
- Wenn immer möglich sollte Zweifaktor Authenfizierung (2FA) zum Einsatz kommen (**Beispiel**: eBanking)

VI. Starke Verschlüsselung und sichere Protokolle

- Daten sollten bei der Übertragung (*data in motion*) und bei der Speicherung (*data at rest*) verschlüsselt werden
- Verwendung der stärksten Verschlüsselung, vorzugsweise IPsec und/oder TLS/SSL
  - ✓ http → https, mqtt → mqtts, ws → wss, etc.
- Bei der Übertragung von unverschlüsselten und unsignierten Daten sollte sichergestellt sein, dass falsche Daten keinen oder nur geringen Schaden anrichten
- Key Management Systeme (KMS) und Key Management Interoperability Protocols (KMIP) verwenden

VII. Minimierung der Gerätebandbreite

- DDoS Attacken können auch mit schlecht gesicherten IoT Geräten durchgeführt werden (**Beispiel**: Mirai botnet)
- Vielen IoT Geräten steht viel zu viel Übertragungskapazität zur Verfügung
- Hersteller sollten den Netzverkehr ihrer Geräte auf ein vernünftiges Niveau limitieren (*kernel-level controls*), um die Geräte für DDoS Attacken unattraktiv zu machen
- Geräte sollten so programmiert sein, dass sie ungewöhnliches Verhalten detektieren und die Werkseinstellungen wiederherstellen
- Leistungsanforderungen jedes Gerätes berücksichtigen und angemessene Einschränkungen vornehmen, welche schwer umgangen werden können

VIII. Sensible Informationen schützen (need to know principle)

- **IoT Grundgedanke:**
  - ✓ Vernetzung von Gegenständen mit dem Internet oder anderen Netzwerken
  - ✓ IoT Geräte bieten Dienstleistungen an (sog. *Services*) und können von anderen Geräten gefunden und verwendet werden
- Oftmals wird zu viel sensible Information (unter anderem *PII*[27] (***P**ersonally **I**dentifiable **I**nformation*) z.B.: Eigentümer, Hostname, etc.) offengelegt, welche mit anderen Informationsquellen verknüpft werden kann, um Attacken durchzuführen
- Nur autorisierte Clients sollten in der Lage sein, Geräte im Netzwerk zu finden

[27] Siehe: Personal data - Wikipedia

IX. Ermutigung für Ethisches Hacken

- Forscher, die Schwachstellen entdecken und berichten, sind der Industrie dienlich
- Hacking für Forschungszwecke sollte der Gesetzgeber daher nicht unterbinden
- Hersteller können mehrfach von einem Bug-Bounty-System[28] profitieren
  - ✓ Milderung schlechter Pressenachrichten
  - ✓ Verbesserung der Produktqualität
  - ✓ Geringere Kosten im Vergleich zu Penetrationstests[29]

X. Einrichtung eines IoT Security und Privacy Certification Boards

- Hersteller müssen die Verantwortung für ihre Kreationen übernehmen bzw. für ihre Kreationen geradestehen
- IEEE und andere Organisationen sollten Zertifizierungsprogramme anbieten
- Zertifizierungsstellen sollten überprüfen, dass
  - ✓ Daten verantwortungsbewusst gehandhabt, geschützt und geteilt werden
  - ✓ Protokolle keine sensiblen Daten offenbaren
  - ✓ Zertifizierte Hersteller auf Datenschutzprobleme umgehend reagieren
  - ✓ Authentifizierung angemessen stark ist
  - ✓ IoT Geräte nicht ungeschützt sind
  - ✓ IoT Geräte ein fälschungssicheres, identifizierendes Etikett haben mit einem Weblink, mit dem Kunden den Zertifizierungsstatus inkl. Gerätebeschreibung finden können

Es empfiehlt sich die Auflistung der *Best Practices für Informationssicherheit* für eine sicherere IT- bzw. IoT-Lösung umzusetzen.

[28] Siehe: Bug-Bounty-Programm – Wikipedia

[29] Siehe: Penetrationstest (Informatik) – Wikipedia

10 Anhang

10.1 SSL/TLS

Die Abkürzung *SSL* bedeutet *Secure Sockets Layer*. Dabei handelt es sich um eine Bibliothek, die sicherstellt, dass mit einer verschlüsselten Verbindung im Internet Daten übertragen werden. In Version 3 wurde die Bibliothek in *TLS* (*Transport Layer Security*) umbenannt. Zunächst findet eine geschützte Authentifizierung und Autorisierung der der Kommunikationspartner statt. Mit Hilfe der asymmetrischer Verschlüsselung erfolgt der Schlüsselaustausch des symmetrischen Schlüssels für die jeweilige Session. Dabei dient der symmetrischen Schlüssel zur Verschlüsselung der Daten.