Advanced System Design

Zusammenfassung

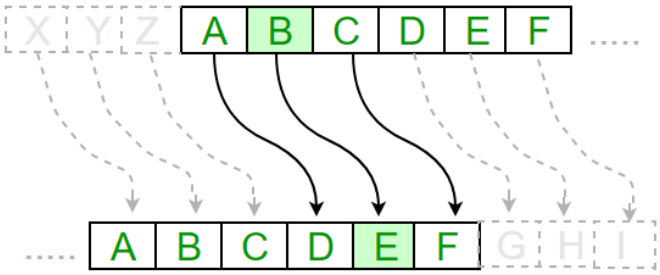
Joel von Rotz / Quelldateien

Inhaltsverzeichnis

Crypto

> **Hinweis**
>
> Wenn Daten Sequenzen in Blöcke geteilt werden (z.B. 64-Bit), dann wird davon ausgegangen, dass bei unvollständigen Blöcken die restlichen Bits mit z.B. `0` augefüllt werden.

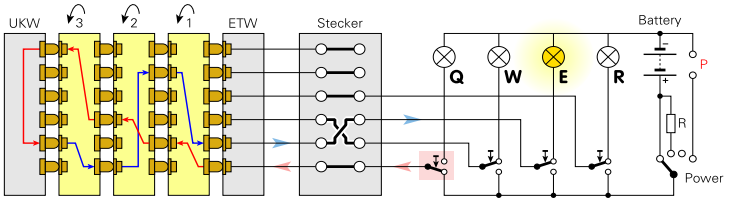
Cesar Cipher / Substitution Cipher



Buchstaben werden um x Positionen verschoben (z.B. $A \xrightarrow{+2} C$). Nachteil ist, dass die Entschlüsselung sehr einfach ist.

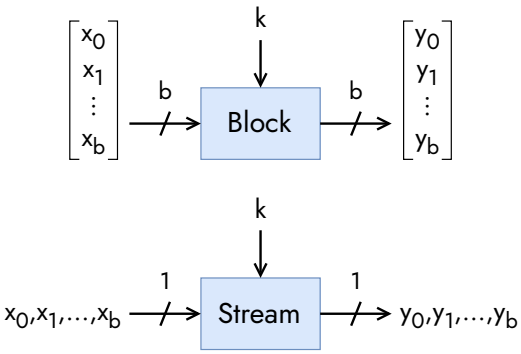
Enigma Maschine

Die Enigma Maschine ist ein komplexes Ent- & Verschlüsselungs System, welches während den Weltkriegen von den Nazis hauptsächlich verwendet wurde (und durch Alan Turing geknackt).



Nachjedem Tastendruck leuchtet ein Buchstabe auf und die Rotoren drehen sich, damit der nächste gleiche Tastendruck nicht den gleichen Buchstabe ergibt. Mit den Steckern können die Buchstaben umkonfiguriert werden (bei Doppelstecker wird z.B. $A \rightarrow B$ & $B \rightarrow A$ und dadurch halbiert sich die Möglichkeiten zu 13).

Stream & Block Cipher



Konfusion

Konfusion ist eine Verschlüsselungsoperation, bei der die **Beziehung zwischen Key und Ciphertext verschleiert** wird. Ein gängiges Element zur Erzielung von Konfusion ist heute die Substitution.

Konfusion erhöht die Mehrdeutigkeit des Ciphertextes und wird sowohl von Block- als auch von Stream-Ciphern verwendet.

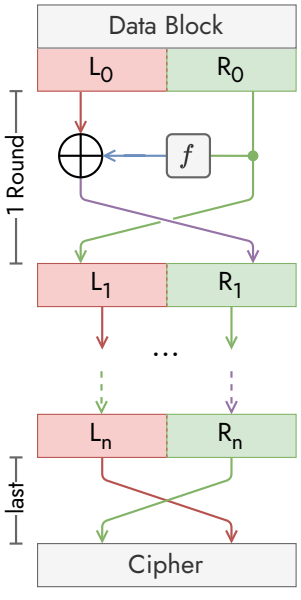
Diffusion

Diffusion ist eine Verschlüsselungsoperation, bei der der Einfluss eines Klartextsymbols auf viele Ciphertext-Symbole verteilt wird, um die statistischen Eigenschaften des Klartextes zu verbergen.

Feistel Network

Ein Feistel Netzwerk wird zum Ver- und Entschlüsseln von Datenpaketen verwendet. Folgend ist ein symmetrisches Feistel Netzwerk → Datenblock wird halbiert (64-Bit → 2 × 32-Bit). Eine Runde entspricht:

$$L_n = R_{n-1}$$
$$R_n = f(L_{n-1}, k_n)$$

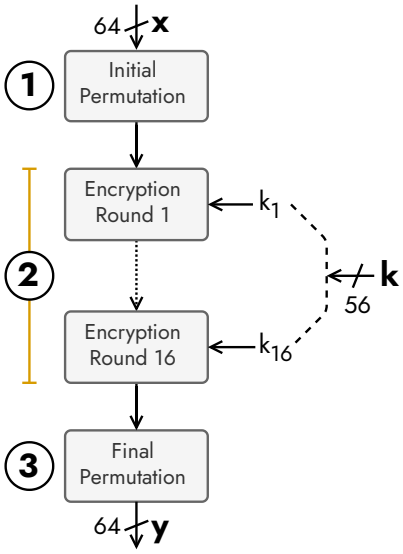


Funktion f ist **wichtig**. Wenn diese sicher gegen Attacken ist, dann wird das Feistel Netzwerk mit jeder Runde und Key-Segment sicherer!

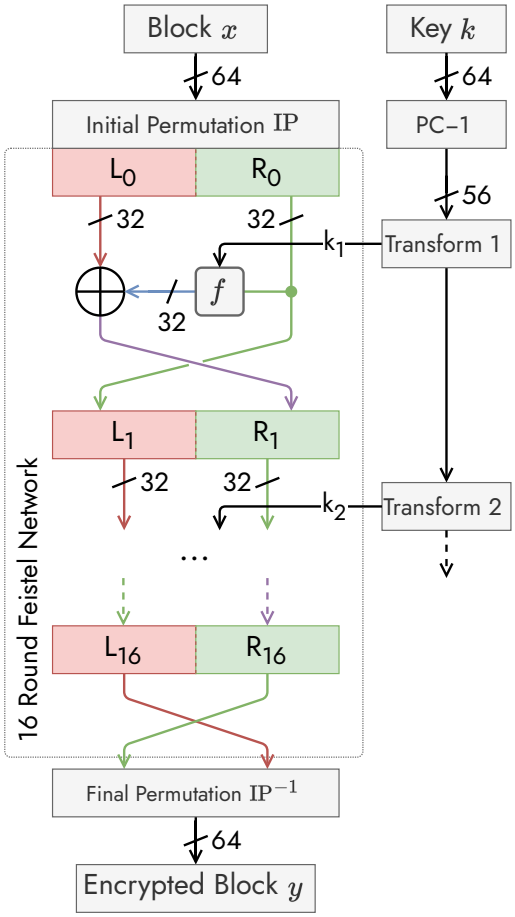
> **!** Ver- & Entschlüsseln
>
> Verschlüsselte Informationen können mit dem genau gleichen Ablauf wieder entschlüsselt werden.

DES

Data **E**ncryption **S**tandard ist ein Cipher, der 64 Bit lange Blöcke mit einem 56 Bit langen Schlüssel verschlüsselt.



1. Data x wird mit einer *Initial Permutation* transponiert (**Diffusion**)
2. Diffusierte Data wird wird im Feistel Netzwerk **16×** verschlüsselt
3. Die verschlüsselte Data wird mit einer *Final Permutation* wieder transponiert (**Diffusion**)



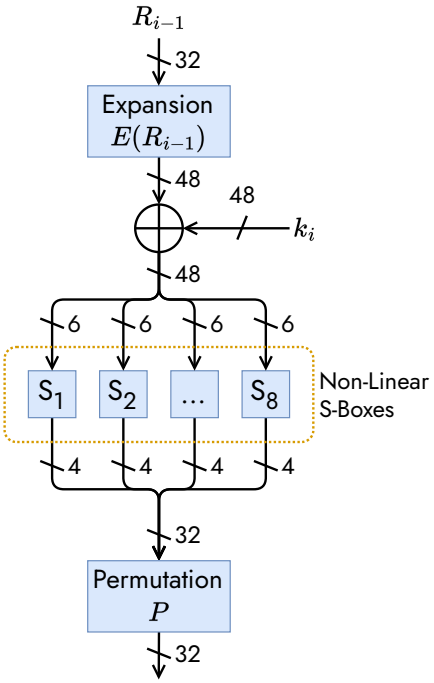
Initial & Final Permutation

Vor und nach dem Feistel Netzwerk werden die Blöcke bitweise permutiert, also wie kreuzverdrahtet (Enigma Steckerbrett) → Bit **Diffusion**

> **i** Hinweis
>
> Diese Permutationen sind in Hardware einfacher implementierbar als in der Software.

f-Funktion

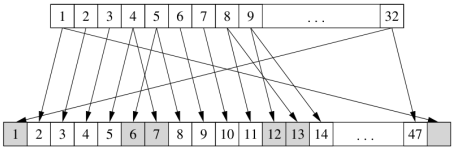
Die f-Funktion vom DES ist das Herz des Algorithmus.



1. Das Datenbyte R_{i-1} wird mit expandiert (Doppelzuweisung) um auf 48 Bit zu kommen...
2. ...danach wird dies mit dem Key ver**xor**t und...
3. ...mit der jeweiligen Substitutationsbox (LUT) S_x verarbeitet.
4. Schlussendlich

Expansion E

Expansion E ist eine spezielle Permutationsfunktion. Die Expansion wird von 32-Bits zu 48-Bits *expandiert*.

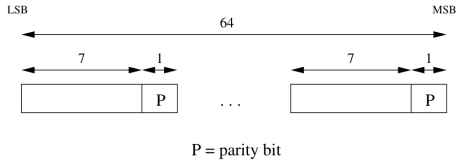


> **!** Wichtig
>
> 16-Bit der 32 Input-Bits kommen doppelt vor. **ABER** ein Input-Bit kommt **nicht** zweimal vor im selben 6-Bit Block → Diffusion wird verbessert, da gewisse Input-Bits zweimal vorkommen.

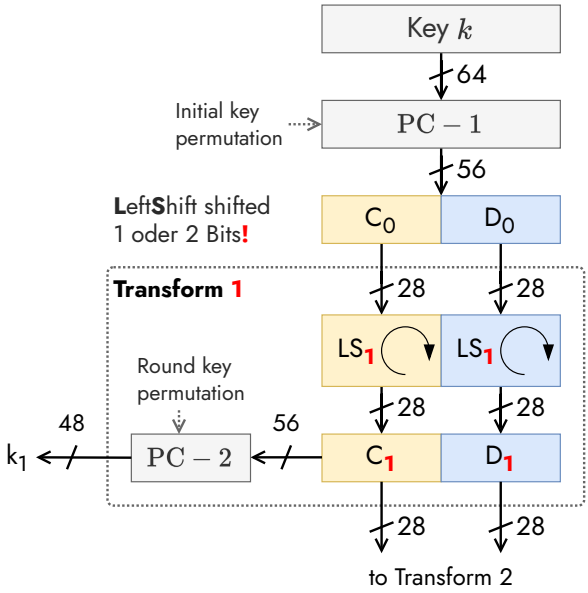
Key Scheduling Transform X

Der *Key Scheduler* generiert 16 Subkeys k_i vom Hauptkey k.

1. Der Key wird in PC − 1 auf 56-Bits gekürzt. Die *Parity* Bits 8, 16, 24, 32, 40, 48, 56 & 64 werden entfernt → sinnlose Bits



2. In Runden $i = 1, 2, 9, 16$ wird C_n **ein** Bit nach links geshiftet, ansonsten **zwei** Bits.
3. In PC − 2 werden erneut 8 Bits verworfen → 48 Bit Subkey k_i



Das Coole

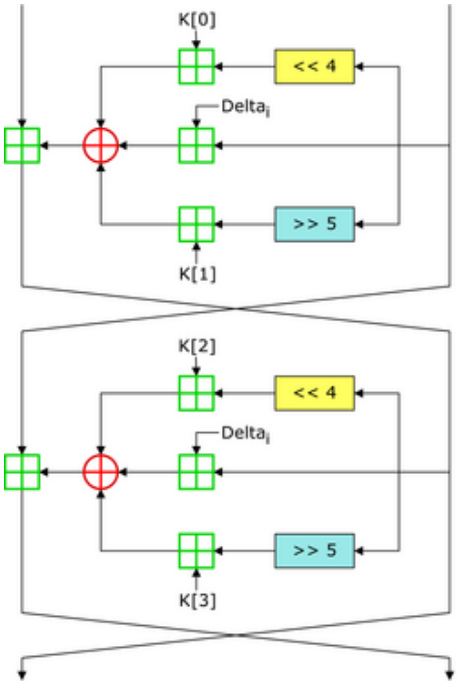
Der Cipher kann mit dem gleichen System wieder entschlüsselt werden. DES verwendet als Grundelement das Feistel Netzwerk, welches diese Eigenschaft hat.

Das Problem

DES ist wegen der kleinen 56 Bit Key nicht sicher (Fall: 1993 & 2008 Brute Force), was ja auch nid so doll is'. Ein Ansatz dafür ist den Key auf 112 Bits zu erweitern durch **triple DES** $(DES(k_1) + DES^{-1}(k_2) + DES(k_2))$.

TEA

Tiny **E**ncryption **A**lgorithm ist ein bereits geknackter und daher auch einfacher Verschlüsselungsalgorithmus. Er verwendet ein *Feistel Network*, 64 Bit Datenblöcke und einen 128 Bit Key. Es werden 32 Runden gemacht, damit die Informationen mehr verschlüsselt werden. Er erfordert sehr wenig Rechenleistung.

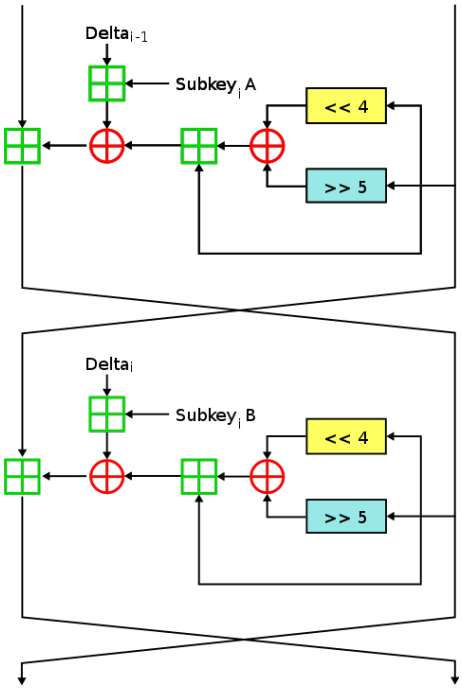


! (Fast) kein Feistel Netzwerk

Die verschlüsselten Blöcke können nicht so einfach wieder entschlüsselt, da im Algorithmus nebst XOR noch Additionen stattfinden. Für die Entschlüsselung werden die **Addition rückgängig** mit Subtraktionen gemacht.

XTEA

e**X**tended **TEA** ist eine Erweiterung von TEA, welcher die Verschlüsselung besser macht. Gleiche Eigenschaften + Konstantwert `delta=0x9E3779B9`.



! XTEA sicher?

XTEA ist ein sicherer Verschlüsselungsalgorithmus, wenn auch nicht so sicher wie RSA oder andere.

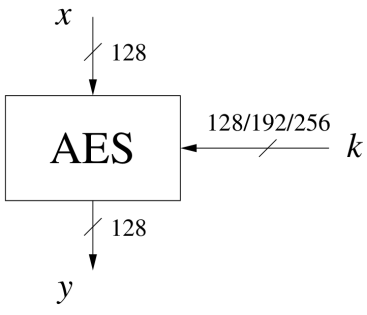
Advanced Encryption Standard (AES)

Die US National Institute of Standards and Technology (NIST) hat in 1977 neuen *Advanced Encryption Standard* (AES) präsentiert und verschiedene Blockcipher wurden evaluiert. *Rijndael* 'gewann' die Runden in 2001 und wurde zu AES umbenannt.

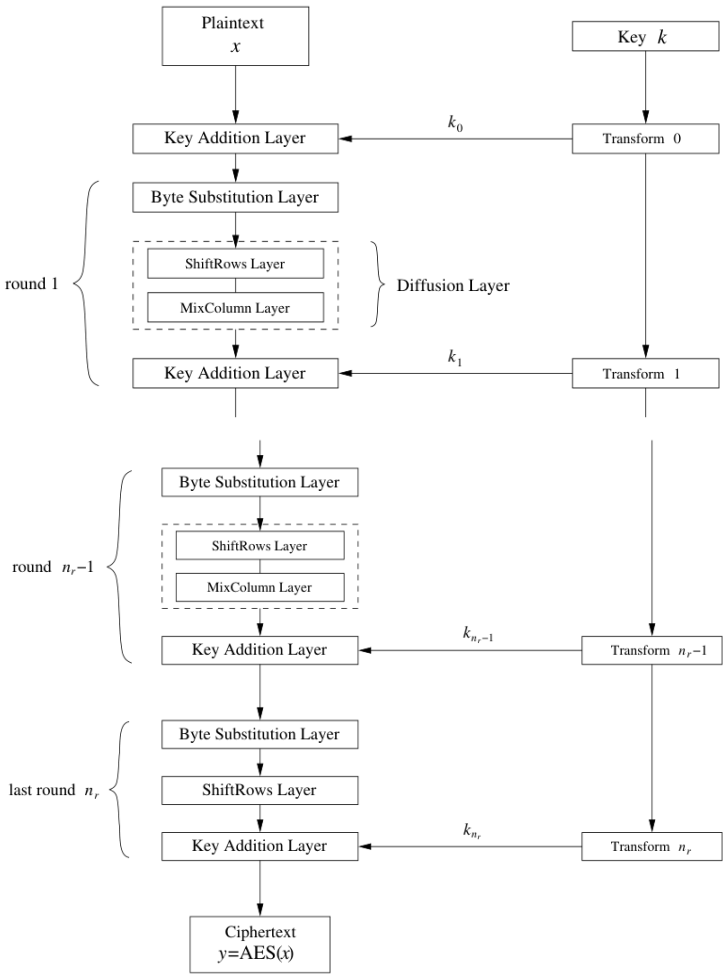
AES ist ein Byte-orientierter Cipher!

i Anforderungen & Kandidaten

Anforderungen - block cipher with 128 bit block size - three key lengths must be supported: 128, 192 and 256 bit - security relative to other submitted algorithms - efficiency in software and hardware
Kandidaten - *Mars* by IBM Corporation - *RC6* by RSA Laboratories - *Rijndael*, by Joan Daemen and Vincent Rijmen - *Serpent*, by Ross Anderson, Eli Biham and Lars Knudsen - *Twofish*, by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson



$k_{128b} \rightarrow n_r = 10$ rounds $\quad k_{192b} \rightarrow 12 \quad k_{256b} \rightarrow 14$



Key Addition Layer

Ein 128b *Round*-Key/Subkey (vom Hauptkey) wird in das Datenpaket ge**xor**ed (Key Whitening)

Byte Substitution layer (S-Box)

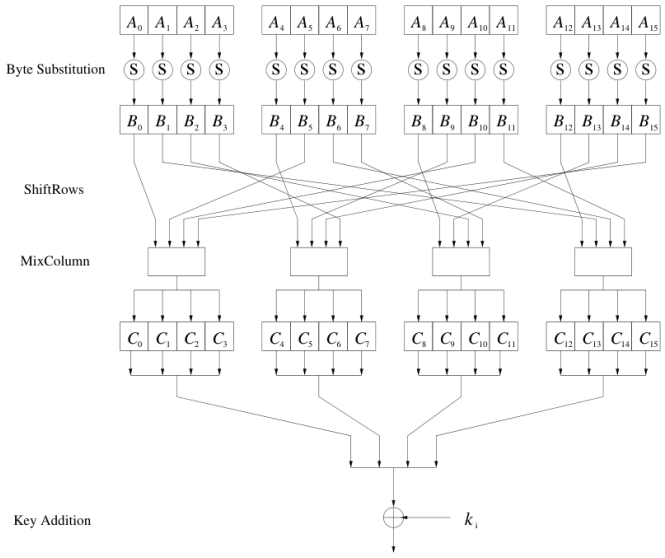
Eine Nichtlineare Datentransformation (Konfusion)

Diffusion layer

Wendet Diffusion an alle Bits an auf zwei Sublayers an. ① *ShiftRows* Layer verändert die Daten auf Byte-Ebene. ② *MixColumn* Layer kombiniert/vermischt 4-Byte-Blöcke via Matrix Operationen.

> **Warnung**
>
> Letzte Runde macht **keinen** MixColumn!

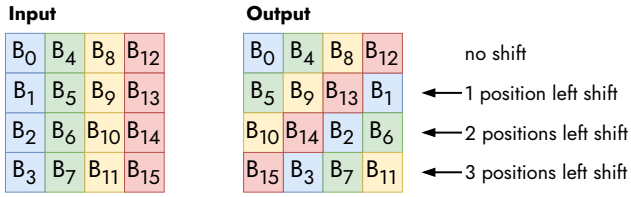
Eine AES Runde für Runden $1, 2, ..., n_r$



Für die Byte Substitution wird folgende S-Box verwendet:

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Shiftrows basiert auf folgendem Verschiebungsmuster.



MixColumns wird mit folgender Matrix-Multiplikation gemacht. Analog kann diese Operation auf die anderen Byte-Gruppen gleich angewendet werden.

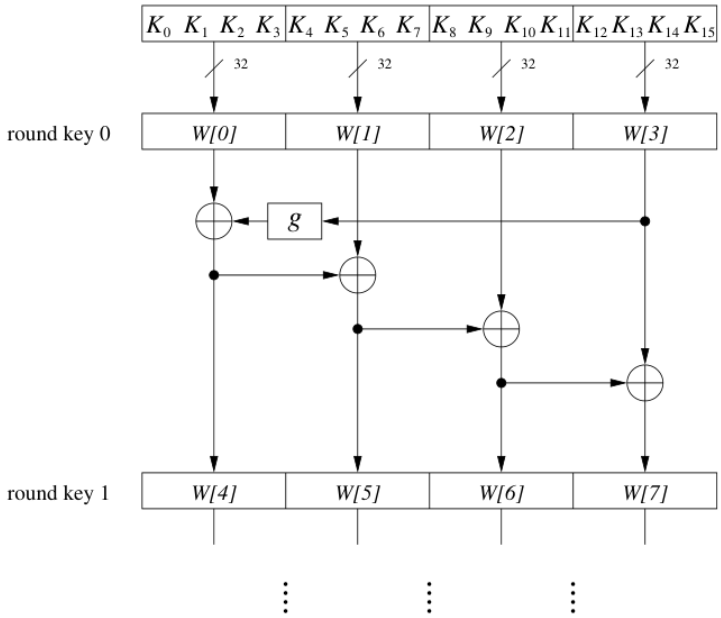
$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{bmatrix}$$

$\mathtt{01}, \mathtt{02}, \mathtt{03}$ sind Representationen eines Elements von $GF(2^8)$. **Addition** wird mit XOR Operationen gemacht (1+1+1=1 - ohne Carry!). **Multiplikation** wird mit Polynom Multiplikation gemacht (① Polynom Multiplikation ② Modulo Operation).

Beispiel mit Input $25_h, 25_h, 25_h, 25_h$

$$\begin{aligned} 01 \cdot 25 &= x^5 + x^2 + 1 \\ 01 \cdot 25 &= x^5 + x^2 + 1 \\ 02 \cdot 25 &= x^6 + x^3 + x \\ 03 \cdot 25 &= x^6 + x^5 + x^3 + x^2 + x + 1 \\ \hline C_i &= x^5 + x^2 + 1, \end{aligned}$$

Key Scheduler / Subkey Generierung

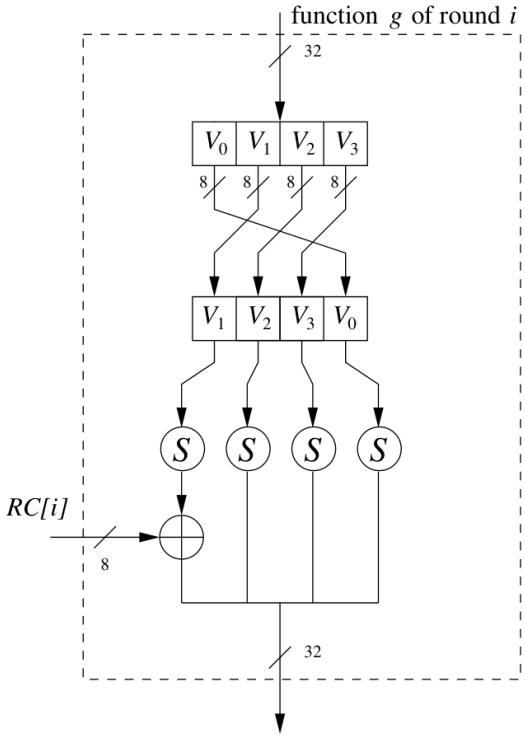


Wie in der Abbildung zu sehen ist, wird das ganz linke *Word* eines Subkeys $W[4i]$, wobei $i = 1, \ldots, 10$ ist, wird berechnet als:

$$W[4i] = W[4(i-1)] + g(W[4i-1])$$

g-Funktion

Die g-Funktion hat zwei Aufgaben. Erstens wird **Nichtlinearität** zum Key Scheduler hinzugefügt. Zweitens wird die **Symmetrie** weggenommen. Beides ist nötig um Block Cipher Attacken zu verhindern.

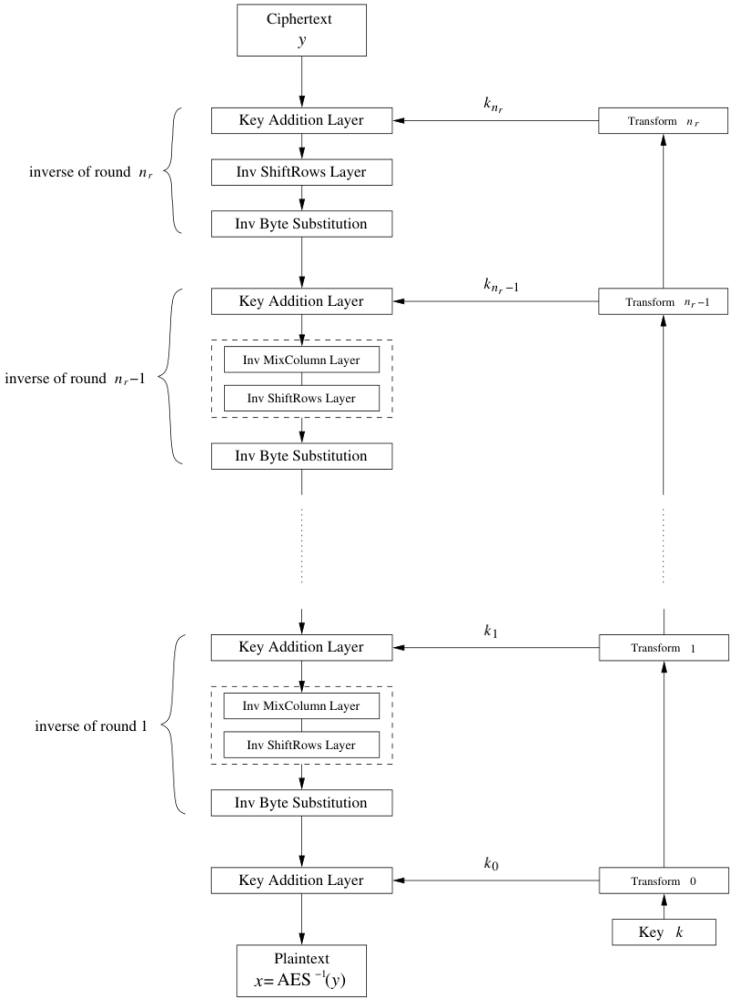


Die g-Funktion rotiert die vier input bytes, führt eine byteweise S-Box substitution und fügt einen Runden-Koeffizient RC dazu. Der Runden-Koeffizient ist ein Element vom Galois field $GF(2^8)$ und wird nur am Byte ganz links dazugefügt. Die Koeffizienten variieren von Runde zu Runde mit folgender Regel:

$$RC[\,] = [(0\times01)_{16}, (0\times02)_{16}, (0\times04)_{16}, (0\times08)_{16}, (0\times10)_{16}, (0\times20)_{16}, (0\times40)_{16}, (0\times80)_{16}, (0\times1B)_{16}, (0\times36)_{16}]$$

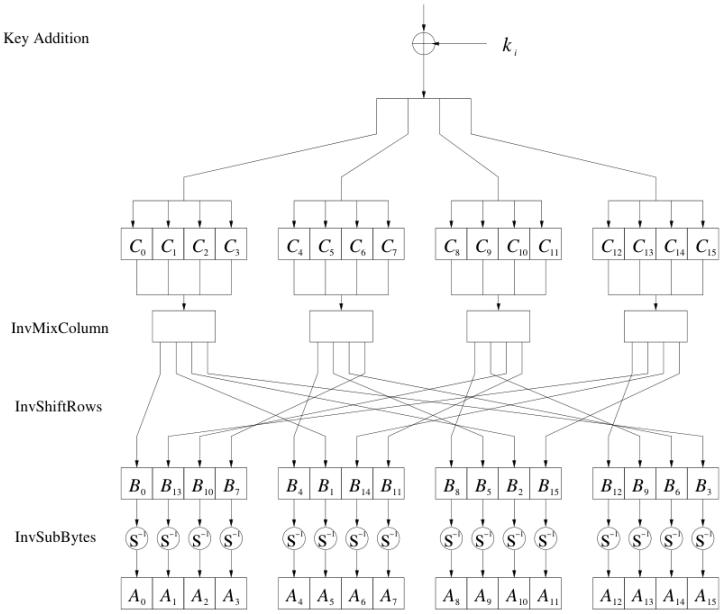
Entschlüsselung

Da AES keine Feistel Netzwerke besitzt, müssen alle Layers invertiert werden, also schrittweise alles Rückwärts machen. Für die Entschlüsselung wird mit angefangen k_{10} (bei 128b) verwendet, dann k_9, ...

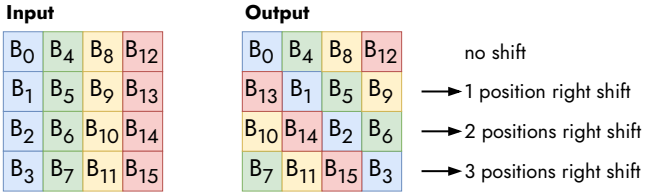


Bei der Verschlüsselung wird zuletzt der MixColumn **nicht** ausgeführt und ist somit analog daselbe bei der Entschlüsselung einfach am Anfang!

Round funktion Die Sublayers werden ebenfalls verkehrt abgelaufen! Die Inverse der MixColumn Sublayer verläuft auf dem gleichen GF-Prinzip, wiederum einfach umgekehrt.



$$\begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

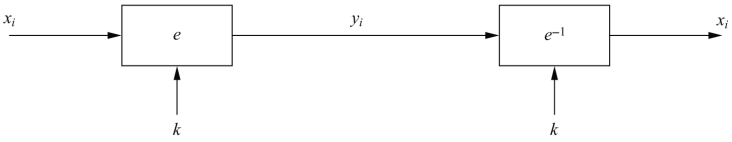


x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
x 8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Cipher Modi

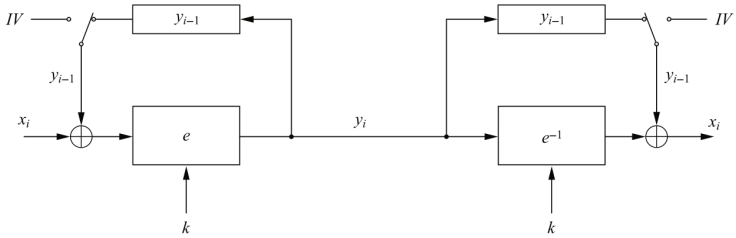
Electronic Code Book (ECB)

Jeder Block wird separat verschlüsselt.



- + No block synchronization required
- + Bit errors only affect the corresponding block
- + Block cipher operating can be **parallelized**
- − Plaintext blocks are encrypted independently of previous blocks
- − Identical plaintexts result in identical ciphertexts → double sending is detectable
- − An attacker may reorder or exchange ciphertext blocks → **Man in the middle attack**

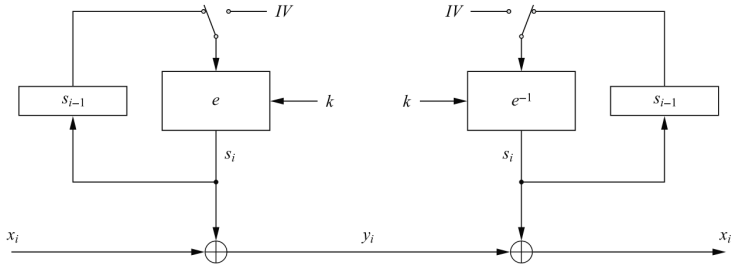
Cipher Block Chaining (CBC)



X_1 wird by einem öffentlich(!) Initialvektor IV (auch *nonce* IV genannt). Alle weiteren Pakete werden mit dem vorherigen Paket verschlüsselt!

- + Ciphertext y_1 depends on plaintext x_1, the key and the IV
- + Ciphertext y_i depends on **all** previous plaintext blocks (and k, IV)
- − No parallelization possible
- − A transmission error destroys **all** following information

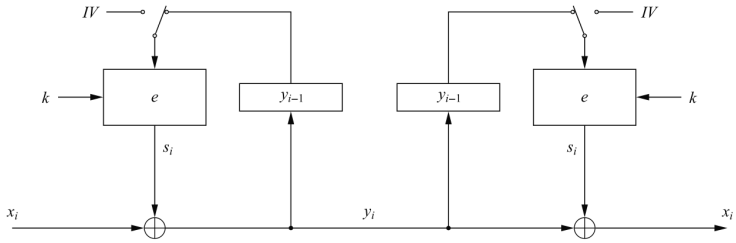
Output Feedback (OFB)



Block Cipher zu synchronem Stream Cipher (**synchron**: Key stream S_i ist nicht vom Cipher selbst abhängig und somit kann der Key Stream komplett vorgerechnet werden).

- + Ciphertext y_1 depends on plaintext x_1, k and the nonce IV
- + Encryption and decryption is exactly the same operation
- + Very fast, S_i can be precomputed
- − Ciphertext y_i only depends on plaintext x_i, k (not on previous plaintexts)

Cipher Feedback (CFB)

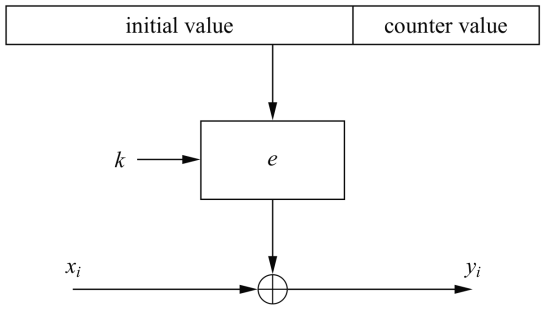


Block Cipher zu **a**synchronem Stream Cipher (**asynchron**: Key stream S_i ist vom Cipher abhängig und ist daher eher langsamer im Vergleich zu OFB).

- + Ciphertext y_i depends on all previous plaintext blocks (and k, IV)
- − No parallelization possible

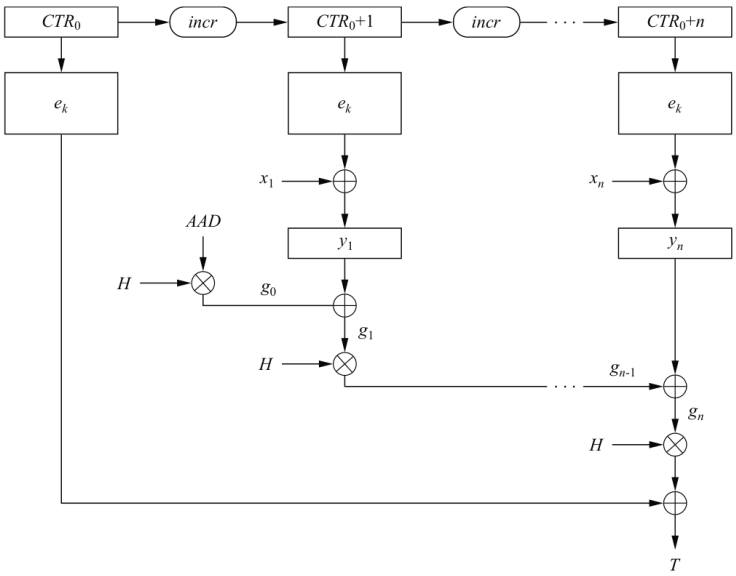
Counter (CTR)

Block Cipher zu synchronem Stream Cipher. IV wird zusammen mit einem Counter zur Key Stream generierung verwendet, was eine Vorberechnung des gesamten Streams erlaubt.



+ Parallelization is possible (no dependance of previous cypher)!
+ Very fast, can be precomputed
− Ciphertext y_i only depends on plaintext x_i, k

Galois Counter Mode (GCM)



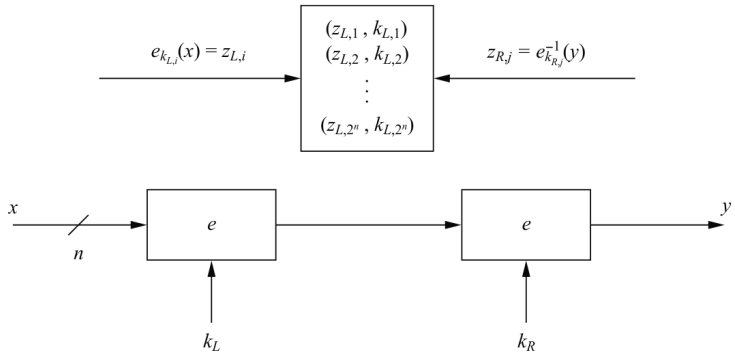
Fügt **message authentication & integrity** als CTR-Erweiterung ein. Alle Multiplikationen werden mit dem 128-Bit Galois Feld $GF(2^128)$ und dem irreduziblen Polynom $P(x) = x^{128} + x^7 + x^2 + x + 1$ gemacht.

+

−

> message authentication & integrity?
>
> **. . . authentication**
> Bob kann prüfen, ob Alice wirklich die Nachricht gesendet hat.
>
> **. . . integrity**
> Bob kann prüfen, dass niemand den Ciphertext während der Übertragung manipuliert hat.

Double Encrpytion & Meet-In-The-Middle Attack (MITM)



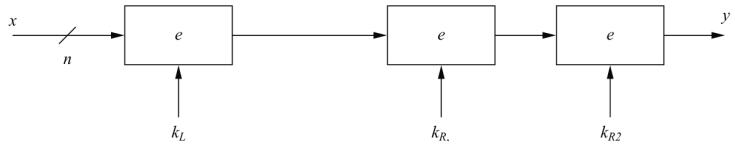
Double Encryption wird der Plaintext x zuerst mit k_L verschlüsselt und dann mit k_R folgend entschlüsselt → Ciphertext y.

Meet-In-The-Middle Attack:

1. **Table Computation** Linke Seite wird via *Brute-Force* den Key k_L ermittelt und daraus ein Lookup Table mit 2^k Einträgen erstellt.
2. **Key Matching** Via *Decryption-Brute-Force* wird die Rechte Seite entschlüsselt und $z_{R,j}$ mit dem entsprechenden Werte von $z_{L,i}$ im Lookuptable verglichen.

Dies reduziert die Komplexizität von 2^{2k} zu 2^{k+1} Suchoperationen.

Triple Encrpytion

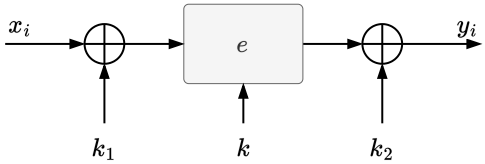


Der Plaintext wird *encrypted-decrypted-encrypted* (EDE) mit drei verschiedenen Keys. Bei einer Key-Grösse von 56-Bits (168-Bit Key komplett) verlängert sich der Aufwand zurück zu 2^{2k} Operationen.

$$y = e_{k1}(e^{-1}_{k2}(e_{k3}(x)))$$

Der MITM-Angriff reduziert die effektive Keygrösse von $3 \cdot 56 = 168$ zu 112 Bits.

Key Whitening



Damit wird DES resistenter gegenüber Brute-Force-Attacken gemacht. Zwei Whitening Keys k_1 & k_2 werden verwendet. Zum Beispiel DESX verwendet Key Whitening.

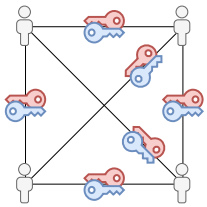
> **Warnung**
>
> Schützt den Blockcipher nicht vor **analytischen** Angriffen wie lineare oder differentiale Kryptoanalyse. Daher ist dieser nutzlos bei schwachen Cipher.

Asymmetrische Kryptographie

Key Distribution Problem

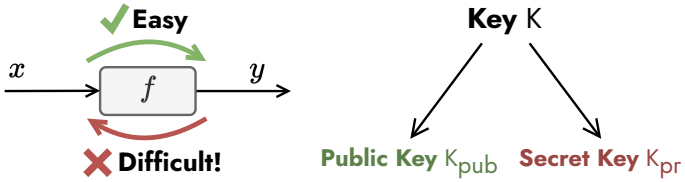
Das Problem der symmetrischen Kryptographie ist, dass jeder Teilnehmer eines Kommunikationsnetzwerkes über die Schlüssel aller anderen Teilnehmer verfügen muss.

$$\#_{keys}(n_{user}) = n_{user} \cdot \frac{n_{user} - 1}{2} \text{ Key Paare}$$

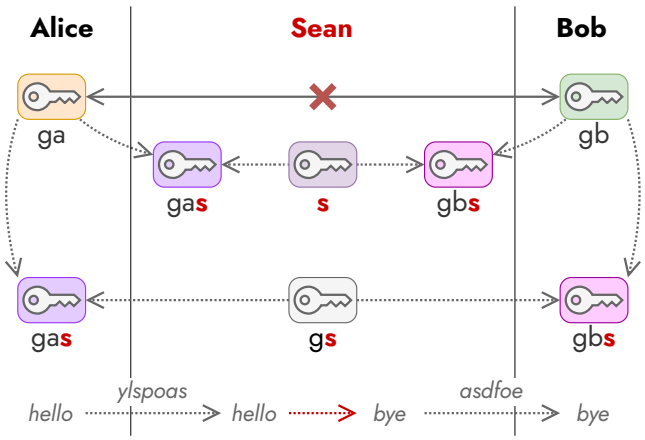


Public Key Cryptography

- Key besteht aus **public** und **private** Teile.
  - **private**: wird zur **Ent**schlüsselung verwendet
  - **public**: wird zur **Ver**schlüsselung verwendet
- Löst das Key Distribution System! **Aber** ist sehr Rechenintensiv!

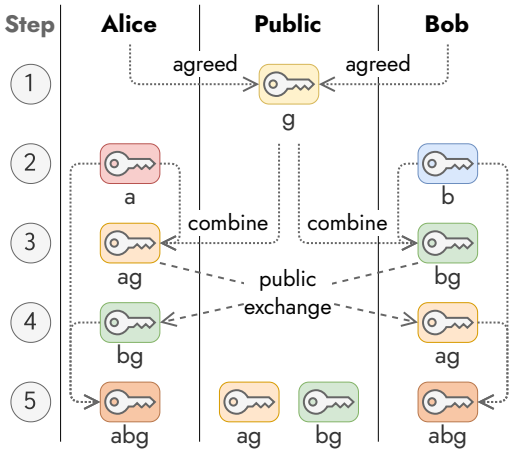


Man-In-The-Middle Attack



Sean (oder Mallory) fängt den Key (ga, gb) Austausch zwischen Alice & Bob ab und antwortet beidseitig mit dem eigenen Key (s→gas, gab). Damit kann Sean jegliche Nachrichten lesen und verändern, ohne dass es Alice & Bob merken.

Diffie Hellman [1], [2]



Vorbereitung:

1. Wählen einer **grossen** Primzahl p
2. Wählen einer **grossen** Ganzzahl $g \in (2, 3, \ldots, n-2)$
3. p & g veröffentlichen

Ausführung:

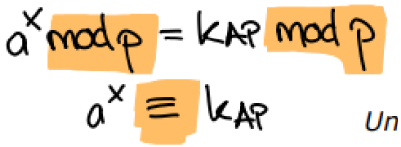
Alice
① Wählen eines zufälligen **private** Key a
② **Public** Key berechnen: $k_A = g^a \bmod p$
③ Key k_A veröffentlichen

Bob
① Wählen eines zufälligen **private** Key b
② **Public** Key berechnen: $k_B = g^b \bmod p$
③ Key k_B veröffentlichen

Alice
Session Key berechnen: $k_{AB} = k_B{}^a \bmod p = g^{ab} \bmod p$

Bob
Session Key berechnen: $k_{AB} = k_A{}^b \bmod p = g^{ab} \bmod p$

Um die Verschlüsselung zu knacken, müsste Oskar (the bad guy) '$g^a \bmod n = k_A$' oder '$g^b \bmod n = k_B$' faktorisieren.



Beispiel

- Set-up by sender Alice or receiver Bob:
  1. Choose a to small prime p=**29**
  2. Choose an integer a ∈ {2,3,..., p−2}=**2**
  3. Publish p = **29** and a = **2**
- Alice: Choose random private key a=**x**,
  Compute public key $k_{AP} = 2^x \bmod 29 = \mathbf{3}$
  Publish public key k_{AP}=**3**
- Bob : Choose random private key b=**y**,
  Compute public key $k_{BP} = 2^y \bmod 29 = \mathbf{7}$
  Publish public key k_{BP}=**7**
- Alice: Compute session key $k_{AB} = 7^x \bmod 29 = ?$
  Use k_{AB} with e.g. AES
- Bob : Compute session key $k_{AB} = 3^y \bmod 29 = ?$
  Use k_{AB} with e.g. AES
- Can you find $2^x \bmod 29 = 3$ or $2^y \bmod 29 = 7$?
- This is called "The Discrete Logarithm Problem" (DLP)
  $\mathbf{x = \log_a k_{AP} \bmod p}$ Find x for $a^x \equiv k_{AP} \bmod p$

!!! Jedes weitere bit in p verdoppelt den nötigen Aufwand um einen Private Key zu finden **!!!**

Elgamal Encryption Scheme

Erweiterung von DHKE → Ähnlicher Ablauf wie bei DHKE, einfach wird ein neuen private Key für jeden Block generiert! Dies **verdoppelt** die Sende-Bandbreite, dafür können die Schlüssel vorgerechnet werden (Im Beispiel auf Alice bezogen)!

- Set-up by sender Alice (or receiver Bob):
  1. Choose a large prime p
  2. Choose an integer a ∈ {2,3, . . . , p−2}
  3. Publish p, a
- Bob : Choose random private key b
  Compute public key $k_{BP} = a^b \bmod p$
  Publish k_{BP}
- Alice: Choose **new** random private **key** a **for every block**!
  Compute public key $k_{AP} = a^a \bmod p$
  Compute session key $k_{AB} = (k_{BP})^a \bmod p$
  Encrypt x with $\mathbf{y = x \cdot k_{AB} \bmod p}$
  Send y and k_{AP} to Bob
- Bob : Compute session key $k_{AB} = (k_{AP})^b \bmod p$
  Decrypt y with $\mathbf{x = y \cdot k_{AB}^{-1} \bmod p}$

RSA

Vorbereitung:

1. **Zwei grosse** Primzahlen $\mathbf{p}$, $\mathbf{q}$
2. Berechne den public Wert $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$ (one way *trapdoor* function)
3. Berechne $\varphi(\mathbf{n}) = (\mathbf{p} - 1) \cdot (\mathbf{q} - 1)$
4. Wählen des public Exponenten $\mathbf{e} \in \{1, 2, \ldots, \varphi(\mathbf{n}) - 1\}$ sodass $\gcd(\mathbf{e}, \varphi(\mathbf{n})) = 1$ (greatest common divisor)
5. private Key $\mathbf{d}$ sodass $(\mathbf{d} \cdot \mathbf{e}) \bmod \varphi(\mathbf{n}) = 1$ gilt

Ausführung:

Alice
Verschlüsselt $y = x^e \bmod n$

Bob
Verschlüsselt $x = y^d \bmod n$

Oskar
Muss Faktor $\mathbf{n}$ aufwenden, um $\mathbf{d}$ zu erhalten

Beispiel

- Setup Bob:
  1. Choose (far too small) primes p = 3 and q = 11
  2. Compute **n** = p*q **= 33**
  3. Φ(n) = (p-1) * (q-1) = (3-1) * (11-1) = 20
  4. Choose **e = 3**
  5. Publish 33,3 ← keine Kommazahl
  6. Private key $d \equiv e^{-1} \equiv 7 \bmod 20$ (e * d mod 20 = 1)
- Alice: Plaintext **x = 4**
  Cryptogram $y = x^e \bmod n = 4^3 \bmod 33 = 64 \bmod 33 = 31$
  Send y=31 to Bob
- Bob : $x = y^d \bmod 33 = 31^7 \bmod 33 = 4$
- Oscar has to factor 33 to get x
- But RSA is malleable: Oskar can change x by $(s^e * y)^d = s^{ed} * x^{ed} = s * x \bmod n$
  => **padding standard** PKCS#1: Add 0x00, random seed, **Hash** and 0x01
- **Side-channel attacks**: Exploit physical leakage of RSA implementation
  e.g., power consumption, EM emanation, etc.

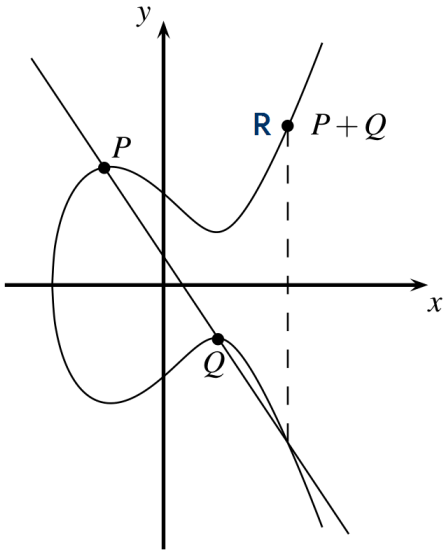
! Side Channel Attacks

Ein *Side Channel Attack* oder Seitenkanalangriff ist eine Sicherheitslücke, die darauf abzielt, Informationen von einem System zu sammeln oder die Programmausführung eines Systems zu beeinflussen, indem indirekte Effekte des Systems oder seiner Hardware gemessen oder ausgenutzt werden, anstatt das Programm oder seinen Code direkt anzugreifen.

Eliptic-Curve-Cryptography (ECC)

Die Eliptic-Curve-Cryptography ist einer vieler Ansätze, um einen Public Key zu generieren. Dieser Ansatz verwendet die Struktur einer eliptischen Kurve, um anhand 'Ketten-Schnittpunkten' einen möglichst lange Kette abzulaufen. Das Endprodukt dabei ist ein kürzerer Key, welcher aber gleichwertige Sicherheit im Vergleich zu Nicht-ECC-Verfahren bietet.

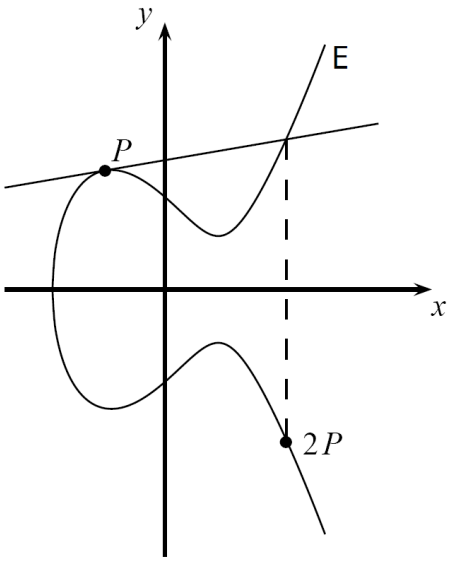
Addition $P \neq Q$



$$R(x_3, y_3) = P(x_1, y_1) + Q(x_2, y_2)$$
$$\text{slope } s = (y_2 - y_1)/(x_2 - x_1) \bmod p$$
$$x_3 = s^2 - x_1 - x_2 \bmod p$$
$$y_3 = s \cdot (x_1 - x_3) - y_1 \bmod p$$

- Linie PQ schneidet die Ellipse E an einem dritten Punkt R′.
- $R = P + Q \rightarrow$ is die Inverse von R′ (gespiegelt an der X-Achse)

Addition $P = Q$



$$R(x_3, y_3) = P(x_1, y_1) + P(x_1, y_1)$$
$$s = \text{tangent} = (3 \cdot x_1^2 + a)/(2 \cdot y_1) \bmod p$$
$$x_3 = s^2 - x_1 - x_2 \bmod p \quad \text{wie bei } P \neq Q$$
$$y_3 = s \cdot (x_1 - x_3) - y_1 \bmod p \quad \text{wie bei } P \neq Q$$

Eliptic-Curve Diffie-Hellmann Key Exchange (ECDH)

- Alice: Choose $k_{APr} = a \in \{2, 3, \ldots, \#E-1\}$ //#E = order
  Compute and publish $k_{PubA} = a*P = (x_A, y_A)$
- Bob: Choose $k_{BPr} = b \in \{2, 3, \ldots, \#E-1\}$
  Compute and publish $k_{BPub} = b*P = (x_B, y_B)$
- Alice: Compute $a* k_{BPub} = T_{ab} = (x_T, y_T)$, define key $k_{AES} = x_T$
  Encrypt $y = AES_{kAES}(x)$
- Bob: Compute $b* k_{PubA} = T_{ab} = (x_T, y_T)$
  Decrypt $x = AES^{-1}_{kAES}(y)$
- Proof for correctness: Alice computes $a*B = a*(b*P) = a*b*P$
  Bob computes $b*A = b*(a*P) = b*a*P$

Security Services

auf English, da ich keine Lust aufs übersetzen habe :-)

1. **Confidentiality**: Information is kept secret from all but authorized parties
2. **Integrity**: Ensures that a message has not been modified in transit
3. **Message authentication**: Ensures that the sender of a message is authentic (An alternative term is data origin authentication)
4. **Non-repudiation**: Ensures that the sender of a message can not deny to be the creation of the message. (e.g. order of a pink car)
5. **Identification/entity authentication**: Establishing and verification of the identity of an entity, e.g. a person, a computer, or a credit card
6. **Access control**: Restricting access to the resources to privileged entities
7. **Availability**: The electronic system is reliably available
8. **Auditing**: Provides evidences about security relevant activities, e.g., by keeping logs about certain events
9. **Physical security**: Providing protection against physical tampering and/or responses to physical tampering attempts
10. **Anonymity**: Providing protection against discovery and misuse of identity

RSA Signature Scheme

Die Schlüssel anhand RSA generieren!

Bob erzeugt die Signatur
"Verschlüsselt" die Nachricht **x** **mit** dem **private Key** **d**
$s = \text{sign}_{K_{Priv}}(x) = x^d \bmod n$
Anhängen der Signatur **s** an die verschlüsselte Nachricht **x**

Alice verifiziert die Signatur
Entschlüsselt die Signatur **mit** Bob's **öffentlichen Key** (**n**,**e**)
$x' = \text{verif}_{K_{Pub}}(s) = s^e \bmod n$

Die Signatur ist gültig wenn $x = x'$

Benötigt Padding: Oscar in der Mitte kann verschiedene & gültige Signaturen generieren, in dem er eine eigene Signatur $s \in Z_n$ wählt und $x = s^e \bmod n$ berechnet. **Jedoch** kann er keine eigene gültige Nachricht erstellen.

Digital Signature Algorithm (DSA)

DSA Key generation:
1. Generate a prime p with $2^{1023} < p < 2^{1024}$
2. Find a prime divisor q of p-**1** with $2^{159} < q < 2^{160}$ //(p-1)/q ∈ N
3. Find an integer α with ord(α)=q
   ord(α) is the smallest positive integer k with $\alpha^k \equiv 1 \bmod q$ // S.93,211
4. Choose a random integer as private key d with 0 < d < q
5. Compute $\beta \equiv \alpha^d \bmod p$
6. Publish $k_{pub} = (p,q,\alpha,\beta)$

DSA signature generation:
Given: message *x*, private key d and public key (p,q,α,β)
1. Choose an integer as random ephemeral (flüchtig) key k_E with $0 < k_E < q$
2. Compute $r \equiv (\alpha^{kE} \bmod p) \bmod q$
3. Compute $s \equiv (SHA(x)+d \cdot r)\, k_E^{-1} \bmod q$ =>
   signature (r,s)

DSA signature verification
Given: message *x*, signature *s* and public key (p,q,α,β)
1. Compute auxiliary value $w \equiv s^{-1} \bmod q$
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$
4. Compute $v \equiv (\alpha^{u1} \cdot \beta^{u2} \bmod p) \bmod q$
5. Valid signature: $v = r \bmod q$;
   invalid signature: $v \neq r \bmod q$

(von Livio gestohlenes) Beispiel

1. Greate a prime p
   With $2^{1023} < p < 2^{1024}$
2. Find a prime q à p-1 mod q = 0
   With $2^{159} < q < 2^{160}$
3. Chose an integer randomly
   With $2 < h < p - 2$
4. Compute $\alpha = h^{\frac{p-1}{q}} mod\ p$
   Check if $\alpha^p mod\ q = 1, h < p$???
5. Choose d randomly as Private Key
   With $0 < d < p$
6. Compute $\beta = \alpha^d mod\ p$
7. Send Public Key = $\{p, q, \alpha, \beta\}$

1. p = 59
2. 59-1 mod q = 0 à q=29
3. 5
4. $\alpha = 5^{\frac{23-1}{11}} mod\ 23 = 5^2 mod\ 23 = 2$
   $3^{59} mod\ 29 = 1$àcheck
5. 7
6. $\beta = 3^7 mod\ 59 = 4$
7. Public Key = {59,29,3,4}

- **Sign Bob**:
  Compute hash of message H(x) = 26
  1. Choose ephemeral key k_E = 10
  2. r = (3^{10} mod 59) mod 29 = 20
  3. s = (26 + 7·20)·3) ≡ 5 mod 29 // $k_E \cdot k_E^{-1} = 1 = 10 \cdot 3 = 30 \equiv 1$ (mod29)
  4. Send (x, r, s))=(x, 20, 5)
- **Verify Alice**:
  1. $w \equiv 5^{-1} \equiv 6$ mod 29 // (5*6) mod 29 = 30 mod 29 = 1
  2. $u_1 \equiv 6 \cdot 26 \equiv 11$ mod 29
  3. $u_2 \equiv 6 \cdot 20 \equiv 4$ mod 29
  4. v = ($3^{11} \cdot 4^4$ mod 59) mod 29 = 20
     $v \equiv r$ mod 29 → valid signature

2. α^p mod q = 1
   $\alpha = h^{(p-1)/q}$ mod p

Hinweis

- Federal US Government standard for digital signatures (DSS) by NIST

Based on Elgamal

➕Signature is only 320 bits long | ➖Slower than RSA signature scheme

Elliptic Curve Digital Signature Algorithm (EC-DSA)

- DSA auf der Grundlage der Elliptischen Kurven-Kryptographie (ECC)
- Bitlängen im Bereich von 160-256 Bit können gewählt werden, um eine Sicherheit zu erreichen, die der von RSA mit 1024-3072 Bit entspricht (symmetrische Sicherheitsstufe 80-128 Bit)
- Eine Signatur besteht aus zwei Punkten, d. h. die Signatur ist doppelt so lang wie die verwendete Bitlänge (d. h. 320-512 Bit für die Sicherheitsstufe 80-128 Bit).
- Die kürzere Bitlänge von ECDSA führt oft zu einer kürzeren Verarbeitungszeit

Key Freshness

Key Freshness... (change keys frequently)

- Limited damage if a key is exposed but was changed often
- Attacks are more difficult if the ciphertext for one key is limited
- Attackers must recover several keys for long pieces of ciphertext ... with Key Derivation
- derive multiple session keys k_{ses} from a given key k_{AB} and a nonce r ("number used only once")

Key Distribution Center (KDC)

Remember the n^2 Key Distribution Problem:
for n participants n (n-1) ≈ n^2 keys are needed (fully connected topology)

KDC is **trusted by all** users

KDC shares an individual **key encryption key (KEK)** with each user =>
Only n long-term key pairs (star topology)

Sends session keys to users which are encrypted with individual KEKs

New users need a secure key (KEK) to communicate with the KDC
(often already in the HW/SW at installation time)

The popular **Kerberos** (authentication and key distribution) uses KDCs

No *Perfect Forward Secrecy*: If the KEKs are compromised, an attacker can decrypt past messages if he has stored the ciphertexts

Single point of failure: The KDC stores all KEKs. If an attacker gets access to this database, all present and past traffic can be decrypted

Communication bottleneck: The KDC is involved in every communication in the entire network (can be countered by giving the session keys a long life time... :-(

Asymmetric Key Distribution (DHKE, RSA, ECDSA...)

Certifying Authority (CA)

Role:
Issues certificates like sigKCA(kpub, ...)

Trust:
Must be trusted by all users.

Certificate:
Contains public key and user ID with CA's digital signature (e.g., $Cert_Alice = (k_{pub}, ID_{Alice}, sig_{KCA}(k_{pub}, ID_{Alice}))$), binding identity to key.

Security Risk:
Vulnerable to man-in-the-middle attacks if the attacker (Oscar) replaces the public key during communication.

Verification:
Requires the CA's public key, which must be distributed over an authenticated channel.

Distribution:
CA public key is typically distributed once during system setup (e.g., pre-installed in web browsers).

Weakness:
Initial transmission of CA public key is not authenticated.

Standard:
X.509 is a common standard with fields like serial number, algorithm, issuer, validity dates, subject, subject's public key, and signature.

Hash Functions

Block

weitere Begriffe

Keyspace Anzahl möglichen & relevanten Keys

Brute Force Alle Keykombination versuchen (Erfolg ≈ Keyspace/2)

Frequency Analysis Gewisse Zeichen(kombinationen) werden häufiger verwendet

Buzzwords

S-Box Substitution-Box , Man in the middle attack , Initialization Vector (IV) , Differential cryptanalysis , NSA No Such Agency ;-) , GCM Galois Counter Mode , Meet-in-the-Middle Attack , TLA Three Letter Acronym , Miller-Rabin Primality-Test , One way trap door function , CFB Cipher Feedback Mode , OFB Output Feedback Mode , Nonlinear $f(A)+f(B) \neq f(A+B)$, Elgamal Encryption Scheme , RSA Rivest Shamir Adleman , DC Differential Cryptanalysis , GCD Greatest Common Divisor , RSA Public Key Cryptography , ECC Elliptic curve cryptography , Probabilistic encryption scheme , ECB Electronic Code Book mode , DSA Digital Signature Algorithm , DLP Discrete Logarithm Problem , CBC Cyber Block Chaining mode , NONCE Number Used Only Once , DES Digitsal Encryption Standard , EEA Extended Euclidian Algorithm , DHKE Diffie–Hellman Key Exchange , AES Advanced Encryption Standard , X-TEA Extended Tiny Encryption Algorithm , EDE encryption-decryption-encryption mode , ECDH Elliptic Curve Diffie-Hellman Key Exchange , EC-DSA Elliptic Curve Digital Signature Algorithm , NIST National Institute of Standards and Technology , Key-Whitening , Confidentiality , Block-Cypher , Prime Fields , Block-Chain , Key-Space , Ciphertext , Puplic-Key , Malleable , Confusion , Integrity , Diffusion , Padding , Enigma , Rounds , Bijectiv , Hash , Traffic analysis attack , Alice / Bob vs Charlie , Fermat Primality-Test , Square-and-Multiply , Side channel attack , CTR Counter Mode , OTP One Time Pad , Double-Encryption , Frequeny-Analysis , Tripple-Encryption , Bute-Force Attack , Symmetric cipher , Cyclic subgroups , Feistel-Network , Vignere-Cipher , Stream-Cipher , Elliptic Curves , Proof of identity , Smart Contracts , Non-repudiation , Double-spending , Central authority , double spending , Birthday paradox , Immutable ledger , Consensus system , Collision resistance , Hirose Construction , Preimage resistance , Proof of elapsed time , Single point

of failure , Distributed consensus , Initial Coin Offering , “proof of work” puzzle , CA Certifying authority , RSA signature scheme , Message authentication , KEK key encryption key , Perfect Forward Secrecy , Davies-Meyer construction , Communication bottleneck , Second preimage resistance , Asymmetric Key Distribution , KDC Key Distribution Centre , Secure Hash Algorithm SHA-1 , Merkle-Dåmgard construction , Miyaguchi–Preneel construction , Matyas-Meyer-Oseas construction , Electronic cash , Key Derivation , Key Freshness , Hash Function , Pseudonymity , Genesis Block , Proof of stake , MD-4 family , Hash-Chain , Immutable , Anonymity , Kerberos , Satoshi , Bitcoin , Mining , X.509

Docker

> **!** Wichtig
>
> Bash Befehle via Host sind mit `$` gekennzeichnet.
>
> ```
> $ echo "this happens on the host"
> ```
>
> Bash Befehle in einem Docker Container sind mit `#` gekennzeichnet.
>
> ```
> # echo "this happens in a Docker container"
> ```

Was'n Docka?

Docker ist eine Plattform zur Software-Virtualisierung mit Fokus auf Wiederverwendbarkeit und Containerisierung.

Die Idee ist, eine Anwendunge mit der nötigen Konfiguration, Runtime und Bibliotheken in ein Paket zusammenzustellen und dann als **portables** Produkt weitergegeben, verarbeitet, etc. ausgeführt werden.

Begriffe

Image
: eine schreibgeschützte und vorgefertigte Vorlage, welche alle nötigen Software and Dateien beinhaltet. Es ist eine "Momentaufnahme" des Filesystems.

Container
: ist die ausgeführte Instanz eines Images und ist eine isolierte Umgebung, welche die entsprechende Prozesse ausführt, **ohne** andere Systeme zu stören.

Dockerfile
: beschreibt wie ein Docker Image zusammengebaut wird anhand Schritten → welche Tools installiert und Dateien kopiert werden

Layers
: Ein Image ist auf Layern aufgebaut, welche während einem Build-Prozess mit einem Dockerfile erstellt werden. Jede Anweisung im Dockerfile erzeugt einen neuen Layer mit einer eindeutigen ID.

Container Layer
: Wenn ein Container kreiert und gestartet wird, entsteht ein Container Layer, welcher alle Änderungen verfolgt. Mit `commit` kann ein neues Image mit diesen Änderungen kreiert werden.

Docker daemon `dockerd` (Service)
: Auch bekannt als *Docker Engine*. Zuständig für Ausführungen der Container und Docker Kommandos.

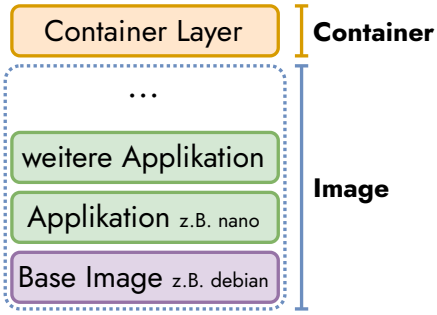
Volumes
: Dateien in Volumes bleiben nach Beendigung/Löschung des Containers erhalten (für Datenaustausch z.B. zwischen Host & Cont. oder seriell/parralel Cont. zu Cont.)

Docker Pfade

- `/var/lib/docker/`: Hauptverzeichnis von Docker
- `/var/lib/docker/images/`: enthält Metadaten zu den Images
- `/var/lib/docker/<overlay-driver>/`: enthält ausgepackte Layer (Images & Container)
- `/var/lib/docker/volumes/`: enthält Volumen

Images

Images sind **schreibgeschützte** Pakete, welches schichtenweise mit Software und Strukturen aufgebaut ist.



>_ Auflisten

Listet alle heruntergeladenen Images auf

```
$ docker images
$ docker image ls
$ docker image list
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
debian       latest    5027089adc4c   3 weeks ago
↪  117MB
```

>_ Löschen

```
$ docker rm <image>
$ docker images rm <Image>
$ docker rmi <image>
```

Löscht ein Image (**WICHTIG** kein Container mit diesem Image sollte dabei existieren, sonst gehts nicht)

```
$ docker image prune
```

löscht *dangling* (nicht gekennzeichnete / Tag = none) Images. -a löscht alle Images, welche nicht verwendet werden

>_ Neue Version veröffentlichen

Nachdem eine Version kreiert wurde, kann diese auf DockerHub veröffentlicht werden.

```
$ docker image push [options] <name>[:tag]
```

>_ Grösse

```
$ docker images
REPOSITORY   TAG      IMAGE ID      CREATED        SIZE
alpine       latest   1d34ffeaf190  2 weeks ago    7.79MB
hello-world  latest   d2c94e258dcb  13 months ago  13.3kB
```

>_ Informationen abrufen

```
$ docker inspect <image>
```

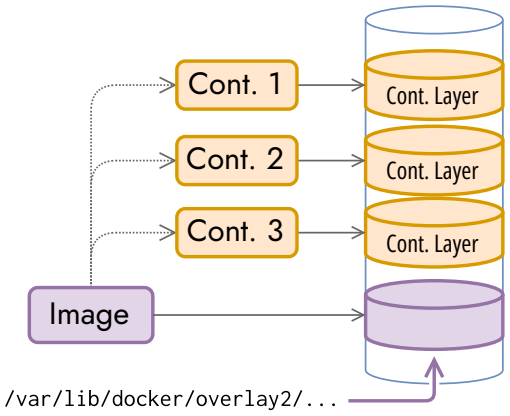
Container

Container sind ausgeführte, **isolierte** Images.

Container-Layer

Da das Image schreibgeschützt ist, werden alle Änderungen, Löschungen und Hinzufügungen am Image in der **Container-Layer** verfolgt → das Image bleibt heil und unversehen.

Dies bedeutet auch, dass das Image nur **einmal** pro Version existiert.



>_ Erstellen/Ausführen/Starten

```
$ docker run <opt> <image> <args>          ①
$ docker create <opt> <image> <args>       ②
$ docker start <container>
```

① Direkt erstellen & ausführen
② Container erstellen und dann ausführen

>_ Auflisten

listet alle **momentan** ausgeführten Container auf. Mit -a listet alle Container auf

```
$ docker ps # kurz für 'docker container ls'
CONTAINER ID  IMAGE   COMMAND ... STATUS        ...
↪ NAMES
79f3f8a71b46  debian  "bash"  ... Up 56 minutes ...
↪ musing_wozniak
6201bc70ef8c  debian  "bash"  ... Up 58 minutes ...
↪ magical_villani
```

>_ Attach/Detach

Ein Container mit einer Shell, z.B. Bash, kann via `attach` oder `-it` zugegriffen werden.

```
docker attach <container>                  ①
docker run -it <image>                     ②
```

① Bei bereits aktiven Container
② Die Shell des Containers wird an den Vordergrund gebracht + Pseudo Terminal

Mit `CTRL+Q CTRL+P` hängt man sich vom Container ab, ohne ihn zu beenden. `CTRL+C` oder im Shell `exit` beendet den Container.

>_ Interaktivität

Ein Container kann auf verschiedene Arten gestartet werden.

```
$ docker start <container>                 ①
$ docker start -i <container>              ②
```

① im Hintergrund
② im Vordergrund

-i/--interactive
: Macht den Container interaktiv und verbindet die Standardeingabe.

-t/--tty
: Aktiviert einen Pseudo-Terminalsimulator für den Container (`...:/# <cmd>`).

```
$ docker run -i debian                     ①
echo hello world
hello world

$ docker run -it debian                    ②
root@containerID:/# echo hello world
hello world

$ docker create -t debian                  ③
$ docker start -i <container>
```

① Interaktiv ohne TTY
② Interaktiv mit TTY
③ In Einzelschritten

>_ Löschen

```
$ docker rm <container> [container ...]
```

Es können nur *stopped* und *created* Container gelöscht werden.

Löschen nach Beendung

```
$ docker run --rm <image>
```

Inaktive Container löschen

Löscht alle Container, welche nicht mehr verwendet werden.

```
$ docker container prune
```

>_ (Um-)benennen

Ein Name kann auf zwei Arten einem Container zugewiesen werden.

```
$ docker rename <old> <new>                ①
$ docker run --name peter_enis debian      ②
```

① Ein bereits existierender Container wird umbenannt
② Bei **Erstellung** eines Containers kann direkt ein Name zugewiesen werden

>_ Dateien kopieren Host ⇄ Container

Der Docker Host kann Dateien in und aus dem Container kopieren.

```
$ docker container cp <container>:<path> <dest>    ①
$ docker cp <src> <container>:<dest>               ②
```

① kopiert dateien vom Container zum Host
② kopiert vom Host zum Container

>_ Ausgabe

Zeigt Log Daten (z.B. Konsolenausgabe) während der Ausführung an (`-f` folgt dem Container/ kontinuierliches Update)

```
$ docker logs <container>
```

>_ Prozesse ausführen/abfragen

Führt Prozesse in einem aktiven Container aus.

```
$ docker exec <container> <cmd>
```

Gibt Prozesse des Containers an (`ax` als Argument gibt alle laufenden Prozesse an)

```
$ docker top <container> <arg>
$ docker container top <container> <arg>
```

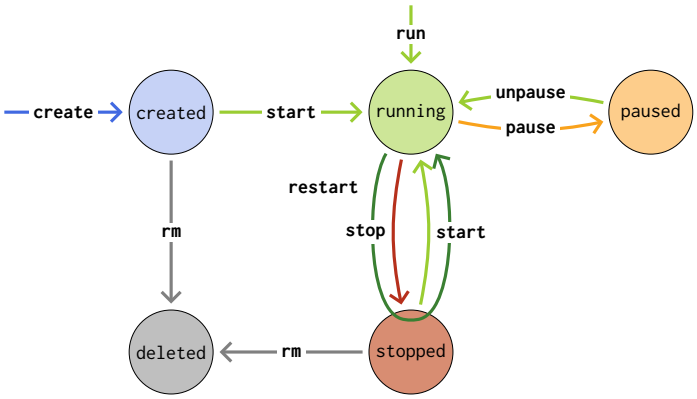
>_ Logging

Zeigt Log Daten während ausführung an (`-f` folgt dem Container/ kontinuierliches update).

```
$ docker logs <container>
```

Lebenszyklus

Ein Docker-**Container** kann fünf Zustände annehmen (Created, Running, Deleted, Stopped und Paused) und kann mit folgenden Docker-Befehlen gesteuert werden.



Unterschied *Stopped* und *Paused*

Wenn ein Container gestoppt wird, werden alle ihm zugewiesenen Ressourcen freigegeben, während bei einem angehaltenen Container kein Speicher, aber die CPU freigegeben wird.

>_ Grösse `--size`

Da ein Container Änderungen auf einer separaten Layer verfolgt, ist der Speicherplatz selbst meistens klein.

```
$ docker ps -a --size
CONTAINER ID   IMAGE         ...  NAMES              SIZE
253be0a7fe55   alpine        ...  brave_pascal       8B
↪ (virtual 7.79MB)
68bb3a7fdf37   hello-world   ...  loving_mcclintock  0B
↪ (virtual 13.3kB)
d20695e9fe4b   alpine        ...  unruffled_tesla    25B
↪ (virtual 7.79MB)
```

`virtual` referenziert auf Container + Image = Totale theoretische Grösse.

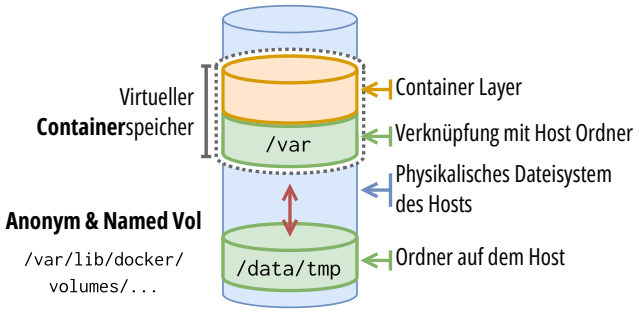
Image kreieren aus Container

Aus einem Container kann ein neues Image erstellt werden:

```
$ docker container commit <container> [image[:tag]]
$ docker commit ...
```

Volume

Container sind voneinander isoliert → Datenaustausch zwischen Host ⇄ Container & Container ⇄ Container wird mit **Volumen** gemacht. Dies muss **explizit** angegeben werden.



Mit dem Parameter `-v` und einem Pfad wird ein Volumen angegeben.

Host Volumes

```
$ docker run -v /path/in/host:/path/in/container
↪  <opt> <image>
```

- `/path/in/host`: Pfad auf dem Host
- `/path/in/container`: Pfad im Container, welche mit dem Host-Pfad verbunden wird.

Named Volumes

```
$ docker run -v name:/path/in/container <opt> <image>
```

- `name`: Name des Volumens
- `/path/in/container`: Pfad im Container, welche geöffnet wird.

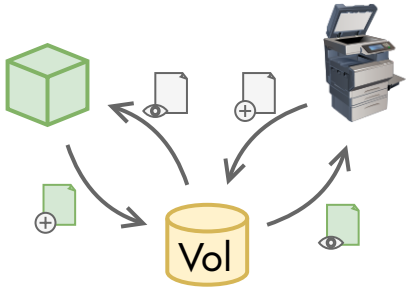
Anonyme Volumes

```
$ docker run -v /path/in/container <opt> <image>
```

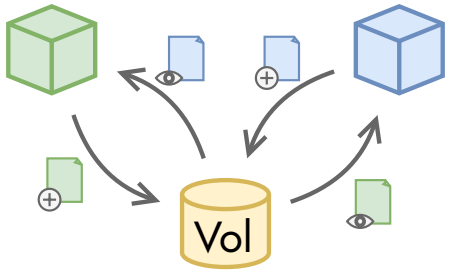
- `/path/in/container`: Pfad im Container, welche geöffnet wird.

Datenaustasch

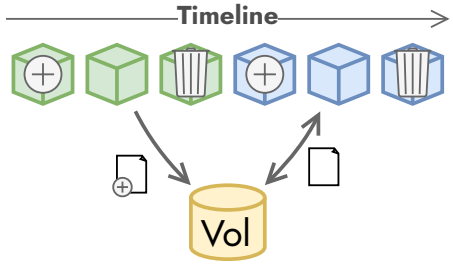
① Host & Container



② Container & Container (parallel)



③ Container nach Container (seriell)



Speicherpfade

Benannte & anonyme Volumen werden unter dem Pfad `/var/lib/docker/volumes/` angelegt.

Host Volumen werden an einem vom Host festgelegten Pfad angelegt.

>_ Auflisten

```
$ docker volume ls
```

>_ Volumen löschen

Um ein Volumen zu löschen, muss der Name des Volumens angegeben werden. Ebenfalls darf es von keinem Container verwendet werden.

```
$ docker volume rm <volume>
```

`prune` löscht nicht gebrauchte anonyme Volumen. Mit `-a/--all` können ungebrauchte anonyme und benannte Volumen gelöscht werden.

```
$ docker volume prune
```

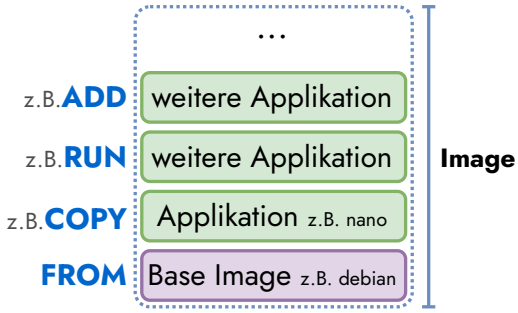
>_ Erstellen

Mit dem `volume create` Befehl können anonyme und benannte Volumen erstellt werden. Einfach keine Pfad/Host Volumen.

```
$ docker volume create [name]
```

Dockerfile

Mit einem `Dockerfile` kann ein Image erstellt werden anhand Anweisungen. Jede Anweisung erzeugt eine eigene Layer.



Anweisungen

```
FROM <image> # image oder 'scratch'
```

Mit der Anweisung `FROM` setzt man das Parent- oder Basis-Image. Es ist die erste Anweisung in einem Dockerfiles. Es können mehrere `FROM` verwendet werden, wenn man Multistage-Building machen möchte.

```
COPY [OPTIONS] <src> ... <dest>
COPY [OPTIONS] ["<src>", ... "<dest>"]
```

Kopiert die Quelle zur Destination. Wenn die Destination ein Ordner ist, können mehrer Dateien kopiert werden. Mit dem Wildcard-Zeichen* und Wildcard-Einzelzeichen ? können mehrere ähnliche Dateien auf einmal kopiert werden.

```
COPY hom*.txt /mydir/
# sammelt alle Dateien z.B. 'home.txt' und 'homie.txt'

COPY hom?.txt /mydir/
# findet 'home.txt', aber nicht 'homie.txt'
```

```
RUN [OPTIONS] <command> ...
RUN [OPTIONS] [ "<command>", ... ]
```

Führt Befehle im Image aus, z.B. `RUN apt-get update && apt-get in`
Mit \ kann der Befehl auf mehrere Zeilen gebrochen werden.

```
ADD [OPTIONS] <src> ... <dest>
ADD [OPTIONS] ["<src>", ... "<dest>"]
```

Ähnlich wie `COPY`, einfach können URLs angegeben werden und gezippte Dateien werden automatisch entpackt.

Startkommando

Die Startkommandos eines Container können auf zwei Arten gesetzt: `CMD` und `ENTRYPOINT`.

```
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT command param1 param2
```

```
# wenn ENTRYPOINT nicht gesetzt ist, wird CMD als
↪  Startkommande verwendet.
CMD ["executable","param1","param2"]

# als default Parameter zu ENTRYPOINT
CMD ["param1","param2"]
```

ENTRYPOINT	CMD	[command]	ausgeführt wird
["script.sh"]			script.sh
["script.sh"]		/bin/bash	script.sh /bin/bash
["script.sh"]	["mysqld"]		script.sh mysqld
["script.sh"]	["mysqld"]	/bin/bash	script.sh /bin/bash
	["/bin/sh"]		/bin/sh
	["/bin/sh"]	/bin/bash	/bin/bash

```
$ docker run [opts] <image> [command] [args...]
$ docker run --rm -it debian ls -la # startet "ls -la"
↪  im Container
```

Um die Startkommandos eines Images anzuzeigen

```
$ docker inspect -f
↪  'ENTRYPOINT:{{.Config.Entrypoint}};
↪  CMD:{{.Config.Cmd}}' <image>
"ENTRYPOINT:[]; CMD:[bash]"

$ docker inspect -f
↪  'ENTRYPOINT:{{.Config.Entrypoint}};
↪  CMD:{{.Config.Cmd}}' kaohslu/01-demo-img
"ENTRYPOINT:[dotnet ASYD_Demo.dll]; CMD:[]"
```

```
WORKDIR /path/to/workdir
```

Mit `WORKDIR` wird der Arbeitspfad gesetzt (ab Aufruf der Anweisung), wo Befehle wie `COPY` oder `ADD` ihre Arbeit verrichten.

Image builden

```
$ docker build <path> -t <name>:<tag>
$ docker build -t <name>:<tag> <path>
```

Image Build History zeigen

Um die History eines Images anzuschauen → Wie das Image erstellt wurde.

```
$ docker history <image>
```

Multistage

Mit Multistage Building können komplexe Images erzeugt werden.

```
# syntax=docker/dockerfile:1
FROM golang:1.21
# FROM golang:1.21 as build
WORKDIR /src
COPY <<EOF ./main.go
package main

import "fmt"

func main() {
  fmt.Println("hello, world")
}
EOF
RUN go build -o /bin/hello ./main.go

FROM scratch
COPY --from=0 /bin/hello /bin/hello
# COPY --from=build /bin/hello /bin/hello
CMD ["/bin/hello"]
```

Netzwerke

Da Container per default isoliert sind (laufen in einem privaten Netzwerk, welches sich Docker kümmert), muss man z.B. Netzwerke **explizit** nach aussen öffnen, wenn man das will.

[TODO?]

Docker Hub

Images suchen & herunterladen

```
$ docker search <searchterm>
```

```
$ docker pull <image>
```

Image Reference Format

Standardmässig werden Images wie z.B. `hello-world` immer vom DockerHub-registry heruntergeladen, aber es ist möglich andere **repos** anzufragen. Es gilt folgendes Format:

`<repo>/<source>/<image>/<tag>`

- `<repo>`: Repository/Content-Host (default `index.docker.io`)
- `<source>` Untergruppe, Hauptprojekt, User, Organisation, etc. (default `library`)
- `<image>` Projekt, wie z.B. eine Runtime

- `<tag>` Version oder Tag des Projektes (default `latest`)

```
$ docker run kaohslu/01-demo-img:latest
```

→ Auf dem offiziellen Repository **DockerHub** wird unter dem User **kaohslu** das Image **01-demo-img** der Version **latest** heruntergeladen und gestartet.

Performance

Cache

Simple 8x8 Cache

A_{8-0}	RAM 512×8
0×000	I0
0×001	I1
0×002	I2
⋮	⋮
0×00E	sub …
0×00F	bmi -0×0E
0×010	⋮
⋮	
0×1FF	

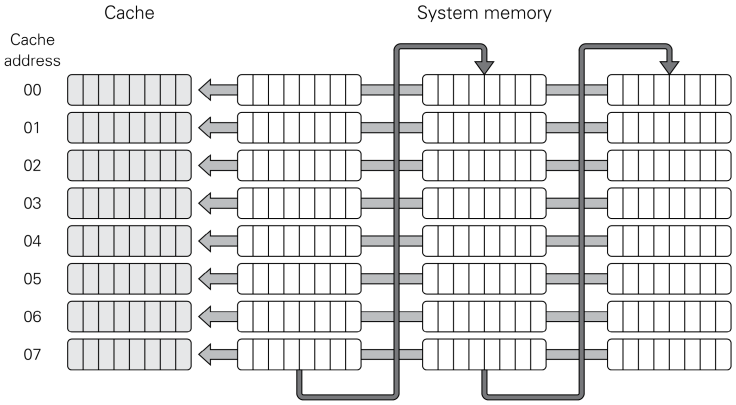
Cache	Tag	A_{2-0}								V
A_{5-3}	A_{8-6}	000_b	001_b	010_b	011_b	100_b	101_b	110_b	111_b	
000_b	000_b	I0	I1	I2	…					1
001_b	000_b						…	sub	bmi	1
010_b	?									0
011_b	?									0
100_b	?									0
101_b	?									0
110_b	?									0
111_b	?									0

valid / invalid

- 8×8 Cache $= 64$ Instr. $\Rightarrow$ 8 cache lines/**pages** containing 8 Instructions and three Tag bits each.
- Valid Bit determines the validity of entry

Tag × Index = Cache Address $\quad A_{2-0}$ = Line Offset

Direct Cache Mapping



Simple & oldest caching system →

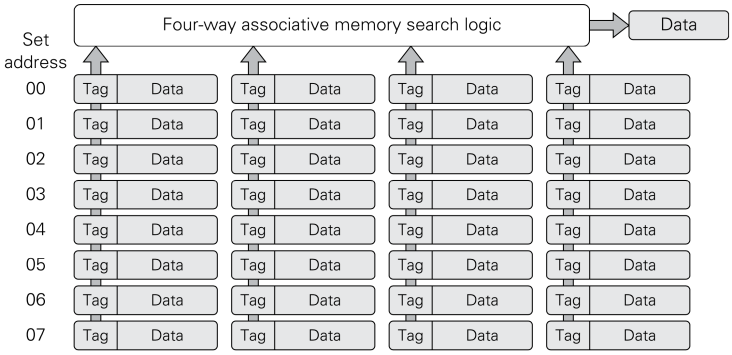
Thrashing / Seitenflattern

A_{8-0}	RAM 512×8
0×000	I0
0×001	I1
0×002	I2
⋮	⋮
0×00E	I14
0×00F	b +50
⋮	⋮
0×041	I16
0×042	sub …
0×043	bmi -66
⋮	⋮

Cache	Tag	A_{2-0}								V
A_{5-3}	A_{8-6}	000_b	001_b	010_b	011_b	100_b	101_b	110_b	111_b	
000_b	~~000_b~~ 001_b	~~I0~~	~~I1~~ I16	~~I2~~ sub	bmi	…				1
001_b	000_b						…	I14	b	1
010_b	?									0
011_b	?									0
100_b	?									0
101_b	?									0
110_b	?									0
111_b	?									0

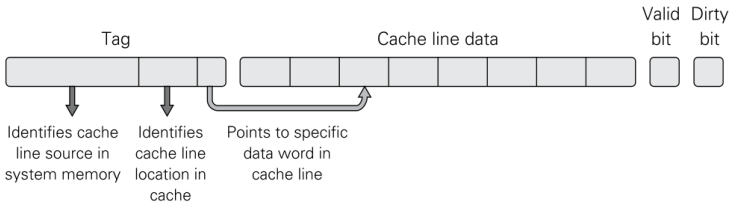
Where main memory is accessed in a pattern that leads to multiple main memory locations competing for the same cache lines, resulting in excessive cache misses. This is most problematic for caches that have low associativity.

Set Associative Cache Mapping



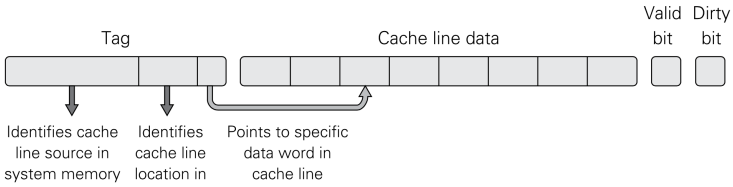
A set-associative caching system reorganises cache lines into sets e.g. with four cache lines per set (*four-way set-associative* cache).

Cache Struktur



Struktur einer Cache-Zeilen

Advanced Cache Lines



Cache Tag Cache line source in system memory

Index Identifies cache line location

Offset Points at specific data word/byte

Valid Bit Set on valid data in cache line

Dirty Bit Set when data has been changed/overwritten by the CPU → write back required

Line Replacement Strategies

FIFO Oldest written line is replaced

LRU (Least Recently Used) the lien with the oldest timestamp is replaced → complex system

Random Cheap and effective, makes thrashing unlikely

NMRU (Not most recently used) randomly, except **not** most recent → performs a bit better than randomly

Valid Bit Set on valid data in cache line

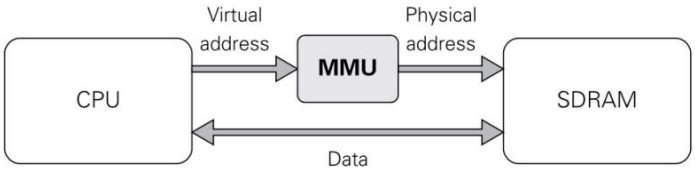
Dirty Bit Set when data has been changed/overwritten by the CPU → write back required

Write Back Strategies

Write-Through Any time a data word in a cache line is changed by the CPU, the cache line is written to memory immediately. This wastes time but the CPU's view of memory is consistent.

Write-Back A "dirty" cache line is written back to memory only when the replacement policy has chosen to evict this line from cache. Write-back avoids a lot of unnecessary system memory writes at the cost of a more relaxed consistency (may cause problems in multi-CPU systems)

Memory Management Unit (MMU)



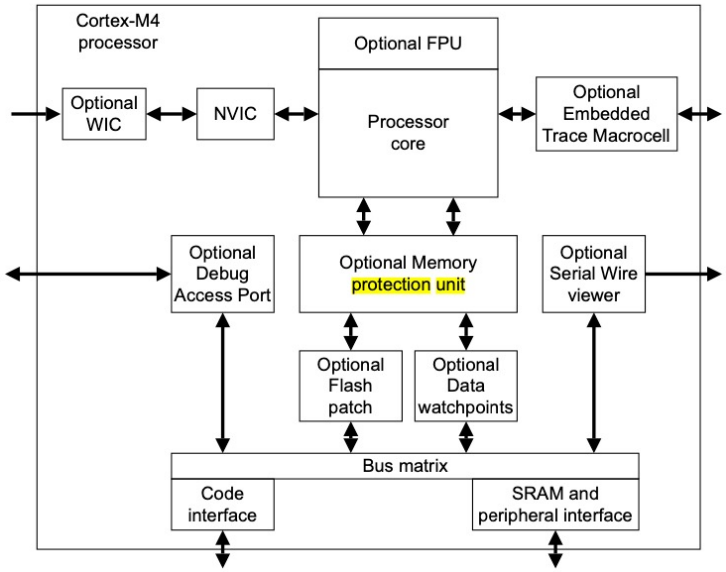
Tasks

Memory Protection: Without an MMU, processes can 'reach' into other processes' memory (unstable & security suffer). Modern systems isolate this by using an MMU.

Virtual Memory: Supports memory swapping, to move infrequently used memory out to a disk and allow high-memory processes to use these areas.

Defragmentation: cleans unused memory caused by long run program

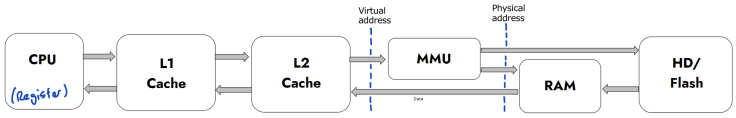
Memory Protection Unit (MPU)



- Separates processes, enforces privilege and access rules to memory
- Prevents task's **stack overflows** into other tasks
- Defineable **No instruction fetch** areas (protect from malicious execution)
- Access type configuration (read-only, write-read)
- Access without permission raises permission fault

MPU on Cortex M4 supports 8 programmable overlapping protection regions, with write, execute bits, cache policy and shareability

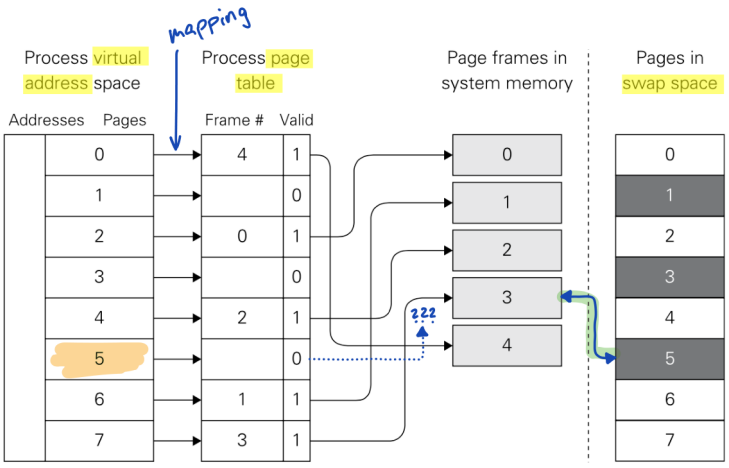
Caching and Virtual Memory



Goal: Give each process the **illusion** that it has its own **private**, ≈ **infinite** system memory

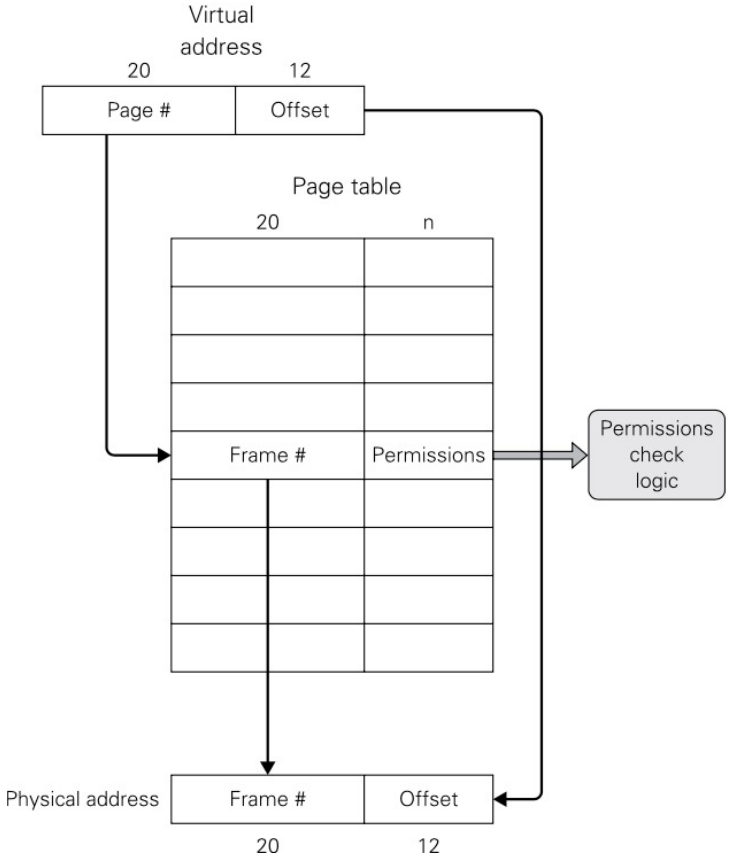
- Is a cooperative venture between the operating system and hardware memory management unit (MMU)
- Virtual address space is divided into many small pages (often 4KB)

Simplified MMU



If **Block 5** is mapped, one of the pages need to be swapped (since no page accesses 5's memory). For example B3 is unused, therefore it is swapped with the page of B5.

Address Translation 32-Bit



- Divides 32-Bit virtual address into 20-bit **page number** and 12-bit **page offset** (2^{12} = 4kB pages)
- Virtual Address → 20-bit lookup in page table + permission check → Physical address is 'returned'/'set'

- Page table entries are usually 4-Byte: $2^{32}/2^{12}$ pages$\times 4^{\text{bytes}/\text{page}} =$ 4MB in size.

Two-Level Page Table

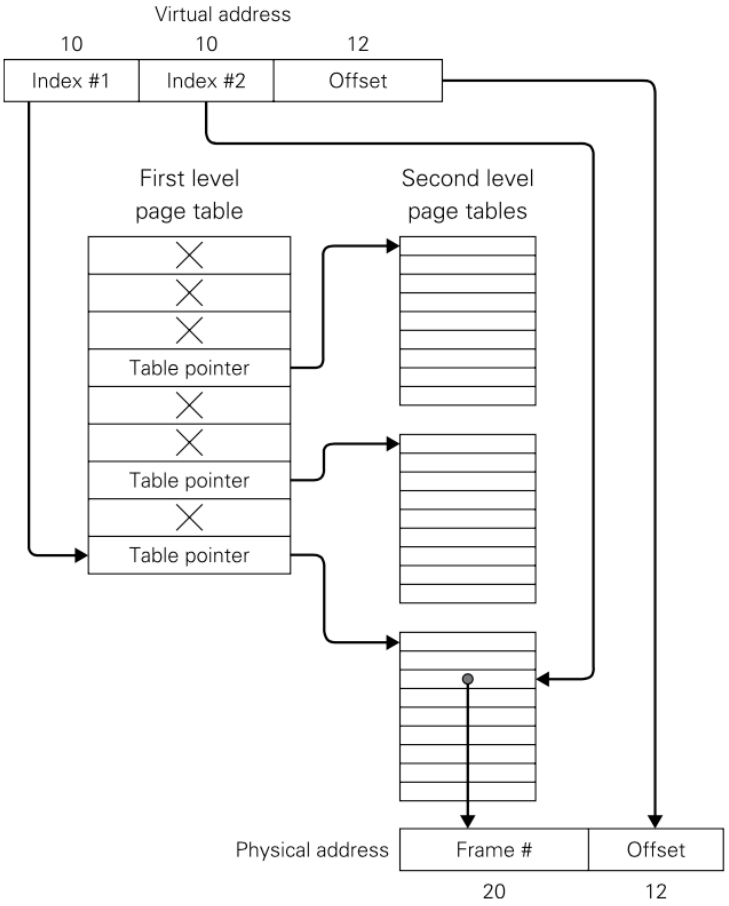
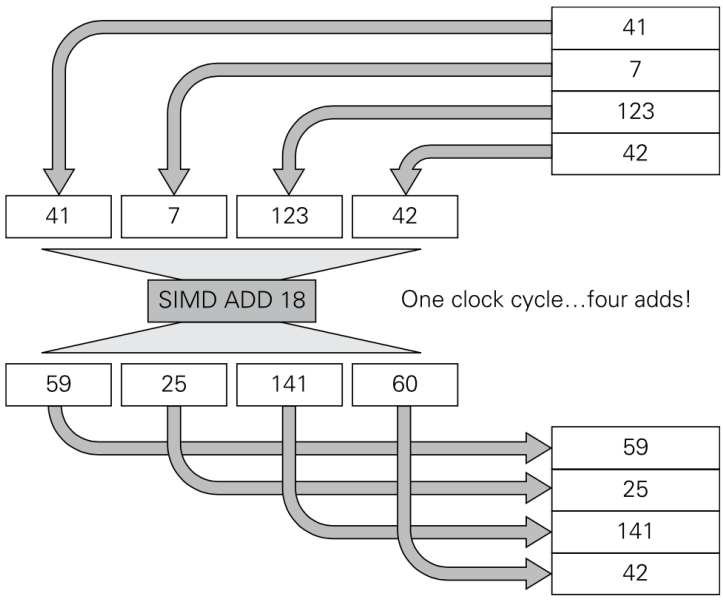


Abbildung 0.1: alt text

- Page table is split up in two
- Does not create unused 4Mb pages

Single-Instruction, Multiple Data (SIMD)

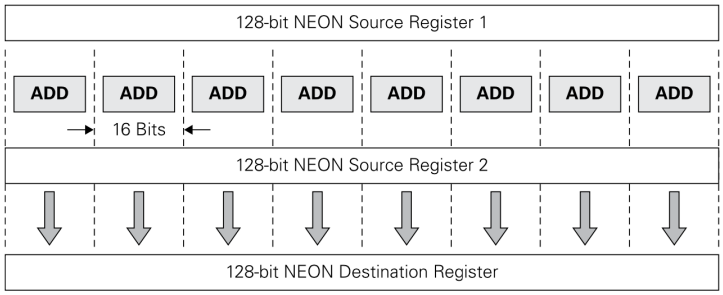
SIMD allows the calculation of multiple data at once via a single instruction (Vector calculations). This is useful for example RGB channel calculations.



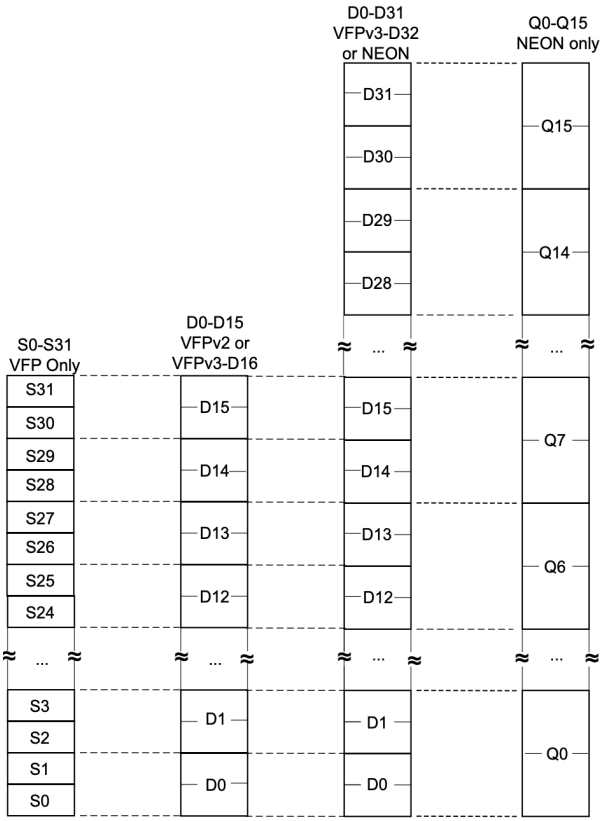
ARM NEON

ARM NEON is a SIMD Coprocessor for the ARM architecture (ARM11, ARMv6 ISA).

- 16 Registers à 128 bits
- Separate pipeline
- >100 SIMD instructions



NEON Registers are 16 × 128bit. **Q**-Registers are 128 bits & **D**-Registers are 64 bits and are mapped to the same Flip-Flops (analogy C/C++ `union`). Vector Floating Point (VFP) **S**-registers use Q0-7/D0-15



Optimization *Loop Unrolling*

von Livio Stadelmann Zusammenfassung :)
Ist eine Compiler-Optimierungstechnik, um die Leistung zu verbessern. Dabei werden Schleifen aufgetrennt und als mehrere aufeinander folgende Schritte aufgeführt, um die Ausführungszeit zu reduzieren. Somit können die Operationen parallel im SIMD durgeführt werden.

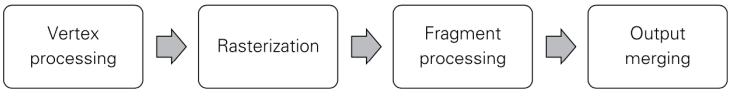
Achtung: Durch das Unrolling wird mehr Speicher benötigt, was zu erhöhtem Risiko von Cache-Miss führen kann.

Graphics Processing Unit (GPU)

GPUs are used to calculate mathematical *recipes*, meaning they can do calculations on repeat really well. Operations such as Additions, Multiplications, Transformations, Comparisons, etc. can be done. An example would be the calculation of graphics for computer games or rendering of movies and models.

The high count of cores (12 Quad Processing Units) and the ability for parallelization makes a GPU suitable for performing many similar calculations. Essentially a calculation recipe is given to the GPU, which applies it to data.

Open Graphics Library (OpenGL)



OpenGL is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering, but can also be implemented on a CPU.

The Raspberry Pi GPU supports OpenGL ES 1.1 and ES 2.0 standards.

Vertex Processing: Vertices define position and shape of an object in 3D space

Rasterization: Converts the 3D space to a (2D) pixel space through analysing the pixel space

Fragment Processing: Series of Operations such as texturing, colouring and blending

Fragment Processing: Fragments are combined to render a 3D object onto a 2D screen.

Modern GPU Hardware

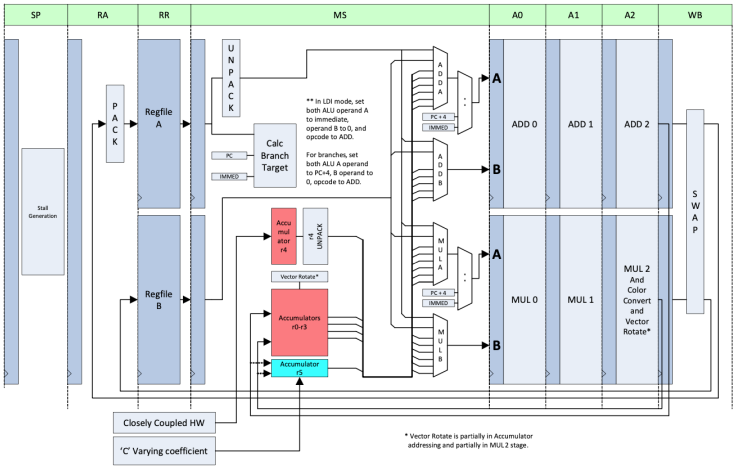
Now comes the question: how can a GPU handle a 1080p, 4 × multi-sampled with 32 bits of colour and depth data of ~60 MB of data, 60fps (2.5 GB/s)?

Through a hierarchy of caches, large and fast DRAM and various specialized DSPs!

VD3: RPi 3 Video Core GPU

- The BCM2836 SoC contains a Video Core IV GPU that supports hardware acceleration of OpenGL ES 1.1 and OpenGL ES 2.0
- The VD3 contains **3 slices** with **4** floating point **shader** (and 2 texture) units each, which contain **4** custom 32-bit floating point **processors** with caches
- **12** Quad Processing Unit (QPU): multithreaded, **16-way SIMD processors**, 32-bit FPU (floating point unit) with **32** general purpose **registers**, two ALU pipelines (addition + multiplication)
- The VD3 does 24 billion floating point operations per second (**24 GFLOPs**)
- Open Vector Graphics (Open VG) for 2D rendering (e.g. Web-browser)
- CPU and GPU have access to a shared region of memory => virtual memory providing the same MMU mappings. **Coherency by** explicitly **flush** caches **or** hardware **broadcast**
- Architectures that make use of compute elements beyond just the CPU (most commonly **CPU and GPU**) are known as **heterogeneous architectures**

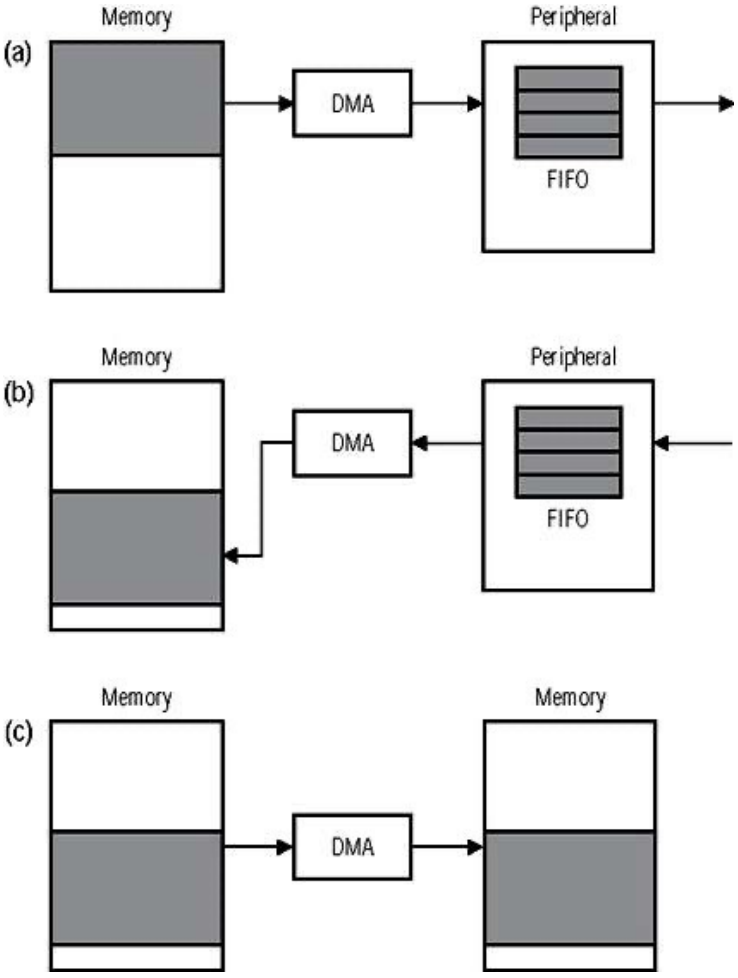
Quad Processor Unit (QPU)



- A and B **regfiles** can address **32** physical **registers** and 32 I/O locations
- Two independent (and asymmetric) ALU units (ADD and MUL)
  - **ADD**: 32 bit or 4*8-bit vector add/subtract with saturation or bit manipulation/shift/logical operations
  - **MUL**: 32 bit or 4*8-bit vector integer or floating point multiply or adds/subs/min/max (in the range [0.0, 1.0]), can convert the **32-bit float result** to scaled 16/8-bit [0.0, 1.0]
- An ALU instruction reads two values, one from register file A and B each, performs an **operation in each** of the add and multiply **unit**s, and writes the results back
- 4-way SIMD processor multiplexed to 16-ways by executing the same instruction for four clock cycles on four different 4-way vectors termed 'quads', icache unit serves four QPUs in four successive clock cycles
- Single **mutex** shared between all QPUs
- **Three** '**delay slots**' **instructions** following a branch are always executed

Direct Memory Access (DMA) Coprocessor

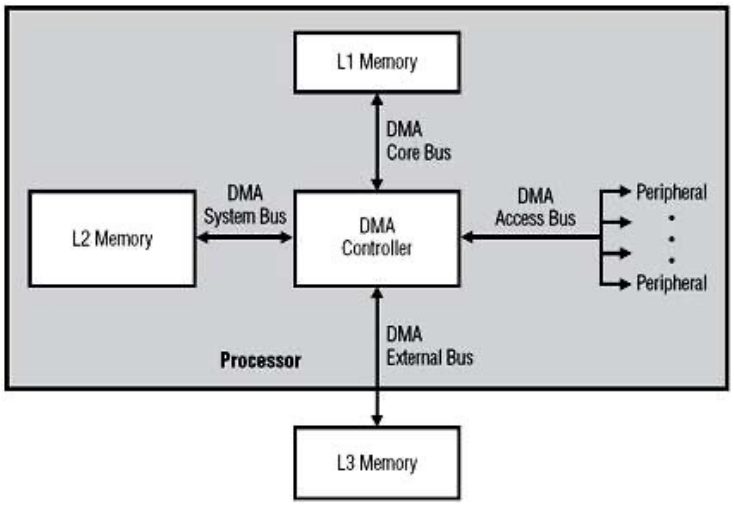
Coprocessor to efficiently move data between memories and/or peripherals independent of the main CPU.



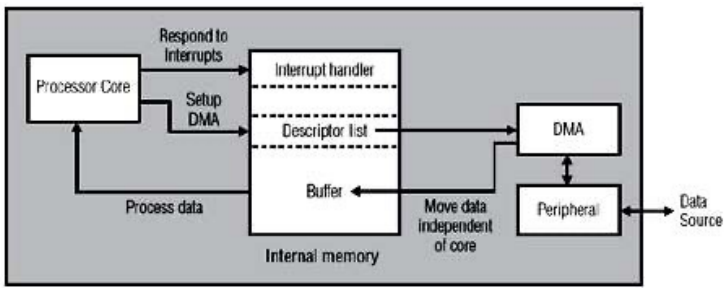
(a) Memory to Interface
(b) Interface to Memory
(c) Memory to Memory

Internal & External DMA Controller

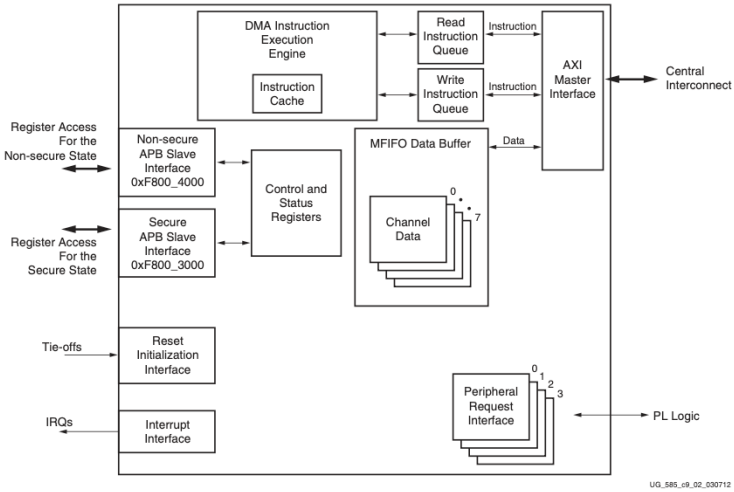
Internal (e.g. STM32 Microcontrollers)



External (e.g. BCM2835 → Raspi)



Block Diagramm



- The DMA engine has a small **instruction set (ISA)**. The instructions are loaded over the AXI master interface from any memory
- 8 DMA **channels** (=threads with round-robin arbitration, equal priority)
- 8 **I-cache** lines of four words (one other channel can be active during cache line loads)
- 8 **Interrupts** (or polling)
- 128 x 64-bit Multi-channel data **FIFO** (MFIFO)
- DMA**LD** and DMA**ST** instructions have a queue of 2x16 word, aggregate data to 64 bit when possible
- Non-/Secure-Mode control and status registers (32-bit **APB** slave interface)

Raspberry Pi BCM2835 DMA

- block-to-block memory transfers and simple peripherals
- must use the **Physical** (hardware) **addresses** of the peripherals
- read only **prefetch** mode to LOAD **into** the **L2 cache** in anticipation of its later use
- **16 DMA channels**

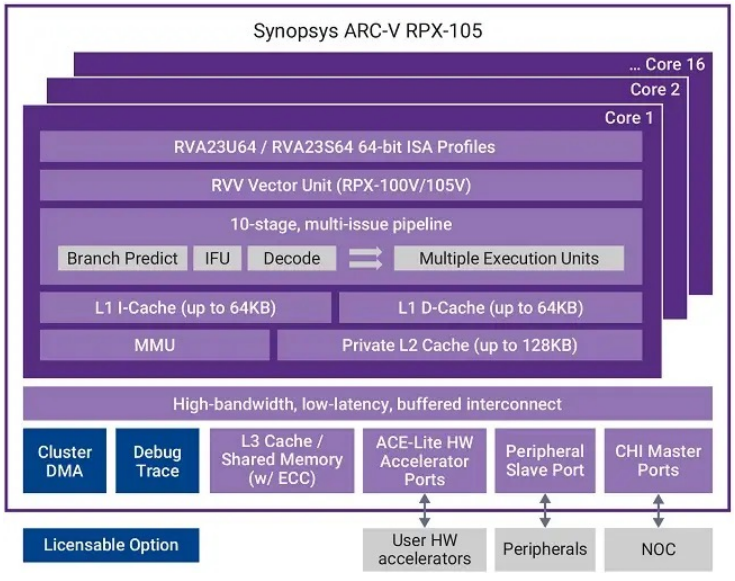
- Each DMA channel loads a **Control Block (CB) data structure** from memory
- A **linked list of Control Blocks** allows to execute a sequence of DMA operations without software intervention
- onto one of the 3 system busses
- bandwidth can be controlled by the bus-arbiter settings
- supports **AXI read bursts** to ensure efficient SDRAM use
- Paced by the peripheral **Data Request (DREQ) signal**
- **Panic signal** to indicate imminent danger of **FIFO** underflow or overflow (sets the AXI apriority level)

DMA Control Block

32-bit Word Offset	Description	Associated Read-Only Register
0	Transfer Information	TI
1	Source Address	SOURCE_AD
2	Destination Address	DEST_AD
3	Transfer Length	TXFR_LEN
4	2D Mode Stride	STRIDE
5	Next Control Block Address	NEXTCONBK
6-7	*Reserved – set to zero.*	*N/A*

Reduced Instruction Set Computer (RISC-V)

RISC



- All instructions have the same size
- One instruction per clock cycle
- Load/Store architecture
- Harvard architecture (instruction + data bus interfaces)
- Many multi purpose register
- Small simple core, big caches

RISC-V

- A high-quality, license-free, royalty-free RISC ISA (Instruction Set Architecture)
- Modular ISA:
  - Small base RV32I (RV64I) **I**=Integer, <50 instructions (x86-ISA 1338 instr. (2015) vs. 80 instr. (1978))
  - Suits all sizes of processor, from smallest microcontroller to largest supercomputer
- **Standard Instruction Extensions**
  - RV32I+**C**: **c**ompressed 16-bit instructions
  - **M**=Integer multiply/divide
  - **A**=Atomic memory operations; **F/D/Q**=Single/Double/Qu precision floating-point; V=Vector...

Register File

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

- **PC** Program Counter outside register file

Integer operations

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

- 47 × fixed-width instructions (32-bit, bit[1-0]=11) + 2 illegal (all 0/1)
- 3 × 5-bit register addresses in fixed locations (read parallel to I-decode)
- 5 instruction formats:
  - **R**: Register-register operations
  - **I/U**: short-lower/long-upper Immediate
  - **S**: Store operation
  - **B**: conditional Branches (S with rotated immediate)
  - **J**: unconditional Jumps (U with rotated immediate)
- I+U together give one 32b constant in any register (also relative in PC)
- Immediate fields are always sign-extended

ALU

The ALU of the RISC-V architecture has **NO FLAGS** (C,O,N;Z)

> 64-Bit Zahl rechnen
>
> R = A + B 64-bit Addition ohne Carry (A in a3, a2; B in a5, a4; R in a1, a0):

```
add a0,a2,a4  # a0=a2+a4
sltu a2,a0,a2 # a0<a2? a2=1:a2=0, calculate
↪  carry in a2
add a5,a3,a5  # a5=a5+a3
add a1,a2,a5  # add carry, a1=a2+a5
```

csr Status and Control Register are 64-bit counter for system ticks, excecuted instructions, productive clock cycles

+C Compressed / 16-Bit Operations

15 14 13	12	11 10	9 8	7 6	5 4	3 2	1 0	
funct4		rd/rs1		rs2			op	CR-type
funct3	imm	rd/rs1		imm			op	CI-type
funct3	imm			rs2			op	CSS-type
funct3	imm				rd’		op	CIW-type
funct3	imm	rs1’		imm	rd’		op	CL-type
funct3	imm	rd’/rs1’		imm	rs2’		op	CS-type
funct3	imm	rs1’		imm			op	CB-type
funct3	offset						op	CJ-type

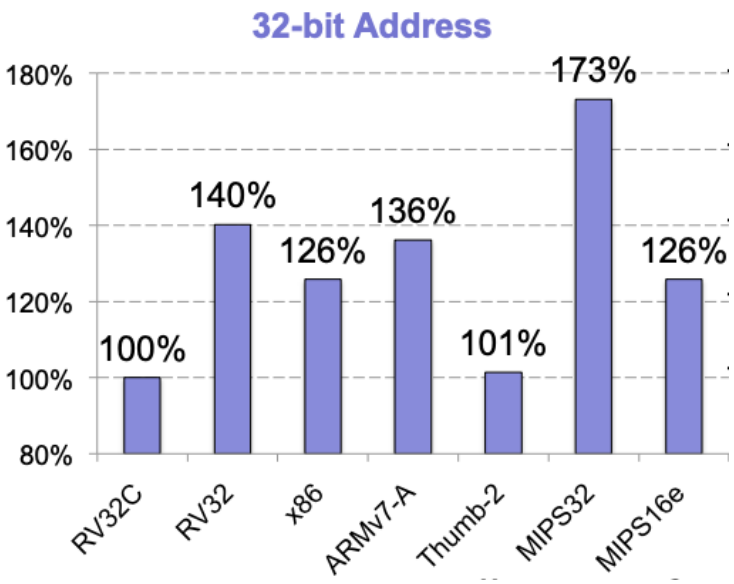
16b decoder is 700 gates (presentation) / 400-8000 gates of the smallest RV32 (book)

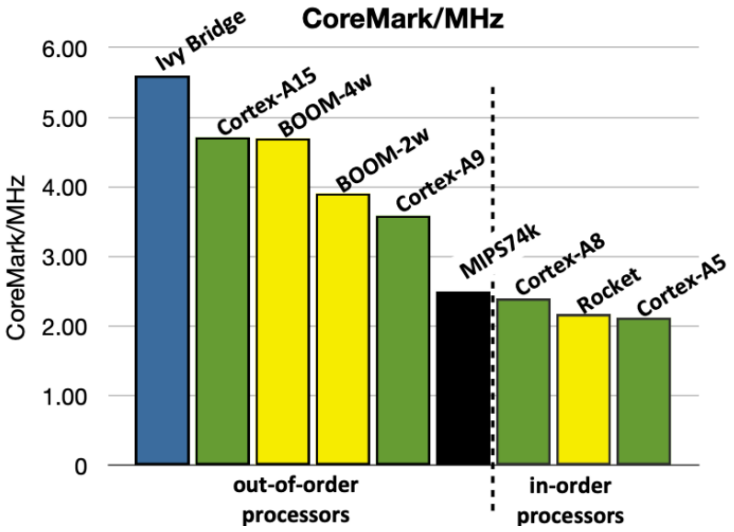
- CIW, CL, CS, CA, CB: rx’ = a0-a5, sp, ra for integers (= s1-s5 for floats)
- Each 16-bit instruction maps to exactly one 32-bit instruction => programmer/compiler does not need to know
- 32-bit instructions can be 16-bit aligned
- 50-60% instructions can be compressed => 25-30% smaller code size

> Illegal Instructions
>
> Following instructions are illegal:

```
#CIW 0...0
0000 0000 0000 0000
#CIW 1...1
1111 1111 1111 1111
```

Code Density & Performance





ARM vs RISC-V

ARM

+ Industry standard, Dense machine code, Large tool base, Large IP base, Barrel shifter in front of ALU, PC in register file, Many address modes, Hard cores on all major technologies, Lots of Tools (e.g. compilers)
− Really expensive (royalties), Large area, Complex design, Complex to understand, Lots of patents

RISC-V

+ Open ISA, Simple HW design, Simple to understand, Small area, Large register file (31*32-bit), Cheap (no royalties), User extendable, No royalties
− HW hard to compile, Code density

ARM

ARM Cortex M: wichtige CPU Register

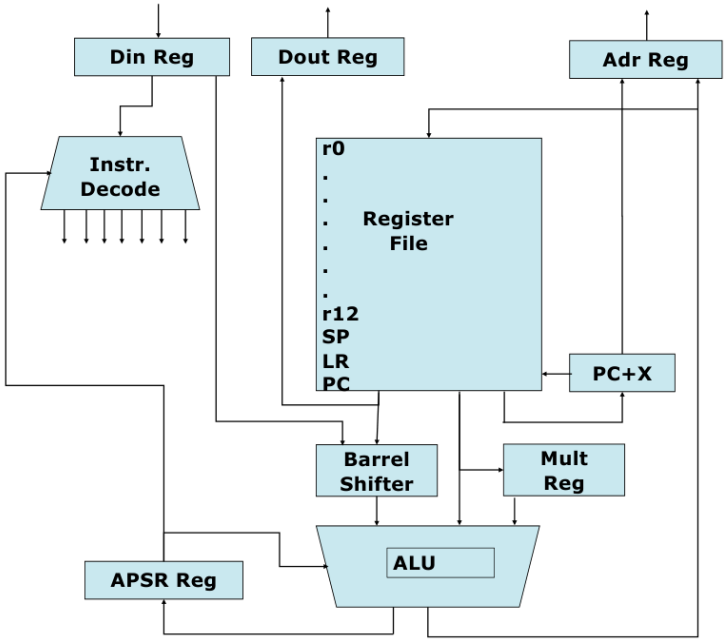
Alle Register sind 32 Bit

APSR:
N bit[31] Negative Flag
Z bit[30] Zero Flag
C bit[29] Carry Flag
V bit[28] Overflow Flag
Q bit[27] Saturation Flag

GE bit[19-16] Single Instruction Multiple Data (SIMD) Flags

APSR	Application Program Status Register NZCVQ+SIMD Flags

r0	General Purpose (Low)
r1	"
r2	"
r3	"
r4	"
r5	"
r6	"
r7	"
r8	General Purpose (High)
r9	"
r10	"
r11	"
r12	"
r13	Main Stack Pointer (MSP)
r14	Link Register (LR)
r15	Program Counter (PC)



32 Bit, nach aussen von Neumann

Reg Register = Speicher (FlipFlop, RAM)
PC Programm Counter
ALU Arithmetic Logic Unit
APSR Condition Code Register
SP Stack Pointer (Stapelspeicher für Kontext und Parameter)
LR Link Register

Safety

Terms

Hazard
A hazard is a situation in which there is actual or potential danger to people or the environment.

Accident
An accident is an unintended event harming people or the environment.

Incident
An incident (or near miss) is an unintended event which does not harm, but has the potential to do so.

Risk To each hazard, the risk describes the likelihood of occurrence and the likely consequences.

Fault
A fault is a defect within the system. Faults can be categorized into random faults and systematic faults.

Error
An error is a deviation from the required operation of the system or subsystem.

System Failure
A system failure occurs when the system fails to perform its required function.

Causalities
The presence of a fault may lead to an error, which may lead to a system failure, which may lead to an accident.

Requirements

Integrity / Dependability

The *integrity / dependability* is a property of a system that justifies placing one's reliance on it.

This demands:

1. *Safety* is a property of a system that it will not endanger human life or the environment.
2. *Reliability* is the probability of a system functioning correctly over a given period of time under a given set of operating conditions.
3. *Availability* describes the probability that a system will be functioning correctly at a given time.
4. *Maintainability* it the ability of a system to be maintained.

V&V&C

Verification
Verification is the process of determining that a system, or module, meets its specification.

Validation
Validation it the process of determining that a system is appropriate for its purpose.

Certification
Certification it the process of convincing a regulatory bodies about a systems properties.

Computers in Safety Related Systems?

Advantages

+ modern digital devices are extremely reliable
+ high speed, low power, small physical size
+ high flexibility, adaptability
+ sophisticated strategies possible (including e.g. diagnostic)

Disadvantages

- complexity, complexity, complexity
- number of possible states to be considered as infinite
- bad predictability due to number of states (i.e. possible failure modes)
- exhaustive testing not possible, detection of failures is unreliable

Silver Bullet

There is good evidence that better processes lead to programs with fewer defect. Some numbers in relation to the CMM (Capability Maturity Model) level defined by Software Engineering Institute (SEI) according to:

CMM Level	Focus	Defects / 1000 LOC
1	None	7.5
2	Project Mngt.	6.2
3	Software Eng.	4.7
4	Quality Processs	2.3
5	Cont. Improvement	1.1

Brooks argues that **"there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity."** He also states that **"we cannot expect ever to see two-fold gains every two years"** in software development, as there is in hardware development (Moore's law).

Hazard Analysis

How to identify the ways in which a system can cause harm?

FMEA: Failure mode and effects analysis

Consider the failure of any component within a system and track the effects of this failure to determine its ultimate consequences.

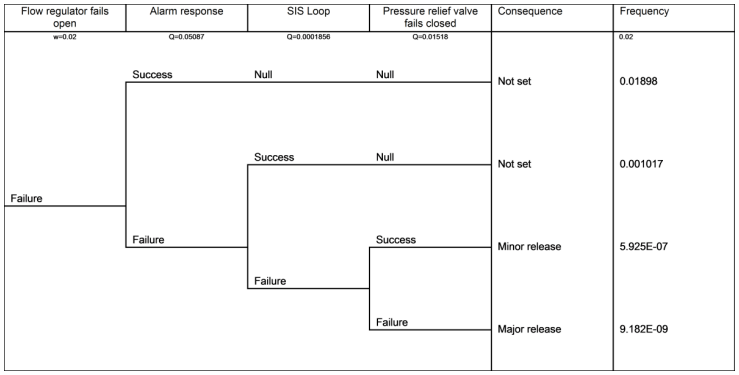
HAZOP: Hazard and operability studies

Use a series of 'guide words' to investigate the effects of deviations from normal operating conditions.

Guide Word	Deviation	Causes	Consequences	Action
NO	No cooling	Cooling water valve malfunction	Temperature increase in reactor	Install high temperature alarm (TAH)
REVERSE	Reverse cooling flow	Failure of water source resulting in backward flow	Less cooling, possible runaway reaction	Install check valve
MORE	More cooling flow	Control valve failure, operator fails to take action on alarm	Too much cooling, reactor cool	Instruct operators on procedures
AS WELL AS	Reactor product in coils	More pressure in reactor	Off-spec product	Check maintenance procedures and schedules
OTHER THAN	Another material besides cooling water	Water source contaminated	May be cooling inefffective and effect on the reaction	If less cooling, TAH will detect. If detected, isolate water source. Back up water source?

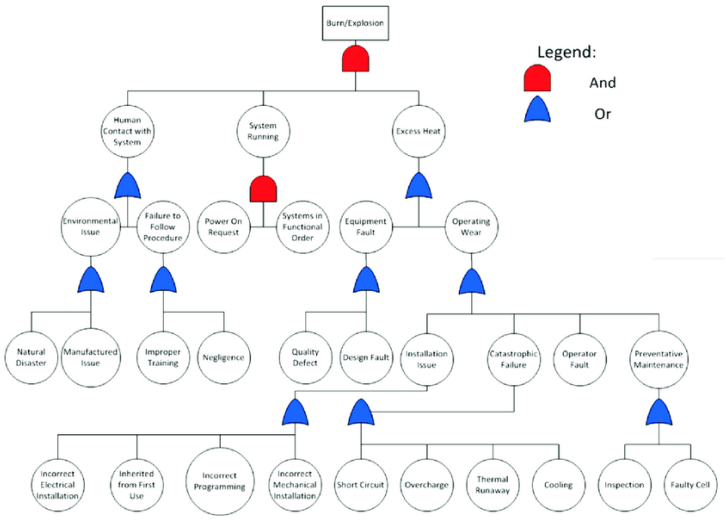
ETA: Event tree analysis

Take the events that can affect the system as starting point and track them forward to determine their possible consequences.



FTA: Fault tree analysis

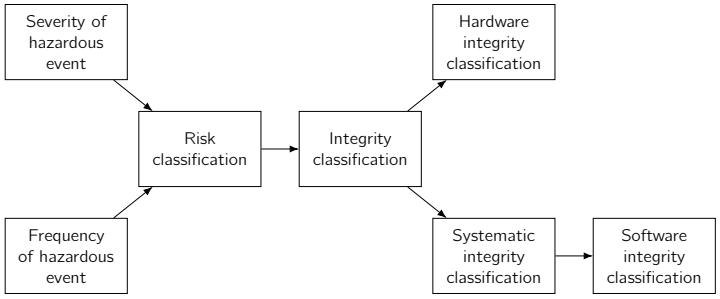
Start with all identified hazards and work backwards to determine their possible causes. (Reverse to ETA)



Risk Analysis

! Fundamental Rule

$$\text{Risk} = \text{Severity} \times \text{Frequency}$$



Severity

How severe is an accident?

Category	Definition
Catastrophic	Multiple deaths
Critical	Single death, and/or multiple severe injuries or severe occupational illnesses
Marginal	Single severe injury or occupational illness, and/or multiple minor injuries or minor occupational illnesses
Negligible	Single minor injury or minor occupational illness at most

Frequency

How frequent does it occur?

Category	Definition	Range (events per hour)
Frequent	Many times in system lifetime	$> 1 \times 10^{-3}$
Probable	Several times in system lifetime	$1 \times 10^{-3} \ldots 1 \times 10^{-4}$
Occasional	Once in system lifetime	$1 \times 10^{-4} \ldots 1 \times 10^{-5}$
Remote	Unlikely in system lifetime	$1 \times 10^{-5} \ldots 1 \times 10^{-6}$
Improbable	Very unlikely to occur	$1 \times 10^{-6} \ldots 1 \times 10^{-7}$
Incredible	Cannot believe that it could occur	$< 1 \times 10^{-7}$

Risk

What is the risk associated?

Frequency	Consequence: Catastrophic	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

I Intolerable
II Undesirable, tolerable only if risk reduction is impracticable
III Tolerable
IV Negligible

Integrity

Is the risk acceptable?

ALARP–Rule

A tolerable risk (class II & III) is acceptable only if it is as low as reasonably practicable.

Risk reduction

Risks can be reduced by means of safety features. The reduction achieved depends upon the integrity of these features.

Safety integrity

Safety integrity is the likelihood of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time.

Integrity Levels

What failure rate is tolerable?

Safety Integrity Level	Continuous mode (prob. of dangerous failure per year)	Demand mode (prob. of failure to perform on demand)
4	$\geq 1\times10^{-5}\ldots1\times10^{-4}$	$\geq 1\times10^{-5}\ldots1\times10^{-4}$
3	$\geq 1\times10^{-4}\ldots1\times10^{-3}$	$\geq 1\times10^{-4}\ldots1\times10^{-3}$
2	$\geq 1\times10^{-3}\ldots1\times10^{-2}$	$\geq 1\times10^{-3}\ldots1\times10^{-2}$
1	$\geq 1\times10^{-2}\ldots1\times10^{-1}$	$\geq 1\times10^{-2}\ldots1\times10^{-1}$

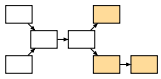
Example – Parachute

Hazard of free fall, severity rated as catastrophic with occasional frequency. Therefore risk is intolerable and a parachute is selected as safety feature. It works on demand mode and has to reduce the frequency by 2 orders of magnitudes, so SIL 2 is required at least.

Distinguish!

Risk is a measure of the likelihood, and the consequences, of a hazardous event. **Safety integrity** is a measure of the likelihood of the safety system correctly performing its task.

Allocating Levels



What is contributing?

Hardware integrity is that part of the safety integrity relating to dangerous **random** hardware failures.

Systematic integrity is that part of the safety integrity relating to dangerous **systematic** failures.

Software integrity is that part of the safety integrity relating to dangerous **software** failures.

Design for Safety

General, Iterative Design Process

1. **Abstraction** Generlization & ID of essentials
2. **Decomposition** Objects into smaller parts + Analysis
3. **Elaboration** detailing & adding features
4. **Decision** identification & selection of alternatives

Types of Fault

What types of fault may occure?

Nature

- Random (HW)
- Systematic
  - Specification
  - HW Design
  - SW Design

Duration

- Permanent – most HW faults, design
- Transient – e.g. α particles
- Intermittent – e.g. contacts, interference (EMC)

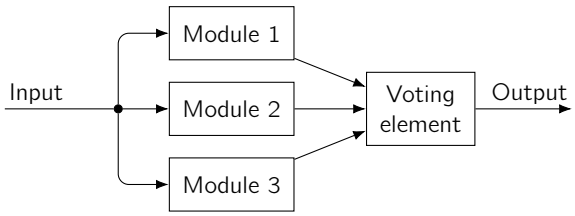
Extend

- Localized – affecting only part of the system
- Global – effects which permeate throughout the system

Fault Tolerance

Triple Modular Redundancy (TMR)

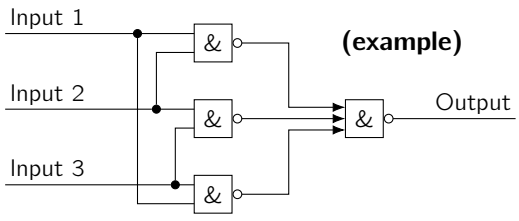
Three identical moduls get fed by the same signal. A voter compares the results and produces an output corresponding to the majority view.



- \+ simple
- \+ prevents from failure of a single component, i.e. *single-point failure*
- − leaves input and voter as sources for single-point failures
- − does not prevent from systematic failures
- − voter is dependable

Voter

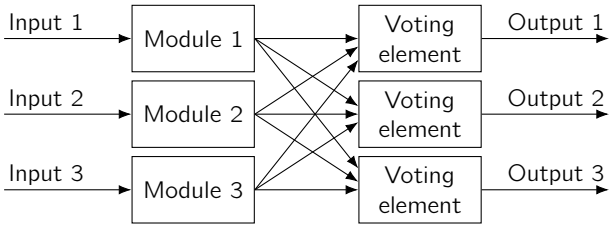
To make voting unit reliable, keep it as simple as possible.



- \+ simple, low complexity
- \+ high reliability
- − no indication in case of discrepancies

TMR with triple voting

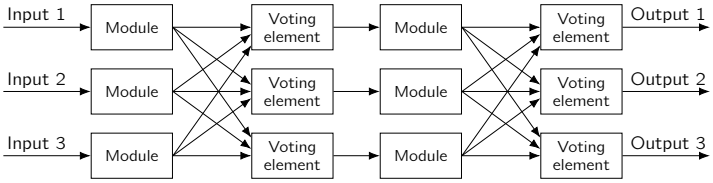
Use triple input signals and triple voting instances.



+ **all** outputs are correct in case a single module fails
− more components required
− no protection against simultaneous failure of two or more modules
− does not prevent from systematic failures

Multistage TMR

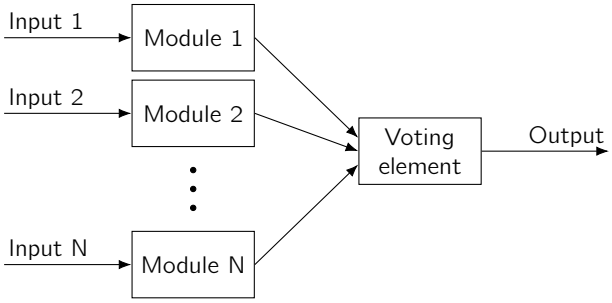
Cascading TMRs with triple voters can deal with failed voting unit.



+ allows a single module to fail at each level
+ allows a single voter to fail at each level
− no protection against simultaneous failure of two or more units at same level
− does not prevent from systematic failure

NMR - N Modular Redundancy

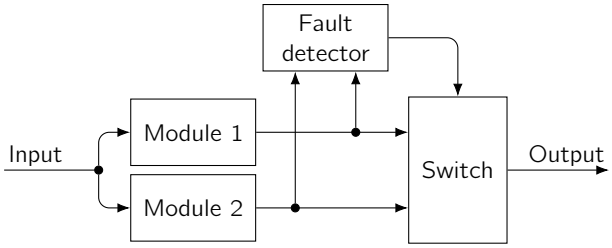
Use a large (odd) number of modules to increase ability to withstand failures.



+ allows $^{(N-1)}/_{2}$ modules to fail
− higher complexity of voter
− higher cost, size, power consumption

Dynamic Redundancy

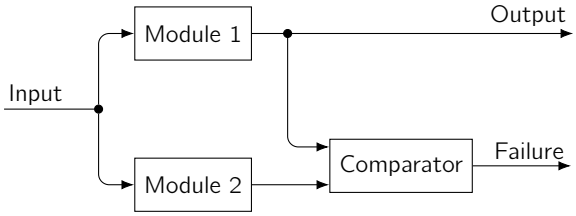
While no fault is detected, one module drives the output. In case of a fault, a switch reconfigures the system such that the output is taken from a 'standby spare' module.



+ allows one modules to fail
+ gives indication of fault
− fault detector required (single-point failure!)
± either *hot standby* or *cold standby* possible
± can be extended to N modules

Self Checking Pair

The output of two identical modules are compared to give an indication of failure.



+ simple, reliable
+ gives indication of fault
− no redundancy

Redundancy Combining

Static Voting to produce *fault masking* at the cost of large amount of redundancy.

Dynamic Fault detection and some form of switching, but **do not** mask faults.

Hybrid Combination of voting, fault detection and module switching → Most reduced down to *N-modular redundancy with spares*.

Software Faults

Software faults are systematic by nature, duplicating the systems gives therefore no protection from faults.

N-Version Programming Same function gets implemented differently with the same specifications. ($N = 3$ for Airbus, $N = 4$ for Space Shuttle)

Recovery Blocks When a module fails, it induces (triggers) the execution of a secondary implementation of the same module.

Reliability

(Un)-Reliability R (Q)

Reliability R is the probability of a component/system functioning correctly over a given period of time and a given set of operating condition.

$$R(t) = \frac{n(t)}{N}$$

N : Amount of identical components
$n(t)$: expected number of components operating correctly at some time t

The **un**reliability Q defines how likely it is that the system/component will break.

$$Q(t) = \frac{n_f(t)}{N} = 1 - R(t)$$

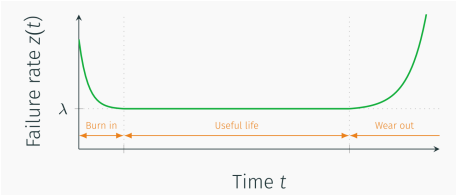
$n_f(t)$: expected number of malfunctioning components at some time t.

Failure Rate $z(t)$

Is the rate at which a device fails. Number of devices failing within a given period of time as a fraction of the devices still functioning.

$$z(t) = \frac{1}{n(t)} \cdot \frac{d\, n_f(t)}{dt}$$

Bathtub Curve



Burn in high 'infant mortality' due to manufacturing faults.
Useful life the failure rate takes in a fairly constant level λ.
Wear out ageing becomes apart and the failure rate rises.

Useful-Life

If the failure rate is constant, $z(t) = \lambda$,

$$z(t) = \lambda = \frac{1}{n(t)} \cdot \frac{d\, n_f(t)}{dt}$$

with $n_f(t) = N - n(t)$ we get the differential equation

$$\lambda n(t) = \frac{dN - n}{dt} = -\frac{dt}{dt}$$

with solution for $R(t) = {}^{n(t)}/_{N}$

$$R(t) = \exp(-\lambda \cdot t)$$

Time-Variant Failure Rates

For software failures, which are systematic and therefore correctable after detection, the failure rate decreases with time. The reliability resulting can by modelled by the *Weibull* distribution:

$$R(t) = \exp(-({}^{t}/_{\eta})^{\beta})$$

β : shape parameter
η : characteristic life

Mean times

Mean Time to Failure

What is the expected time that a system will operate before the first failure occures?

$$MTTF = \int_0^\infty R(t)\ dt \rightarrow \int_0^\infty \exp(-\lambda \cdot t)\ dt = \frac{1}{\lambda}$$

Beware!

A system with $\lambda = 0.001$ failure/h does have a MTTF of 1000 hour. But the reliability at t = 1000 hour is $R(t) = e^{-\lambda \cdot t} = e^{-1} \approx 0.37$. Chances that any given system runs for 1000 hour are only $\approx 37\%$!

Reliability vs. MTTF

Reliability
a function of time, depends on the time for which the system must operate.

MTTF
fixed characteristic that does not change with time.

Mean Time to Repair

What is the average time required to repair a system that has failed?

$$MTTR = \frac{1}{\mu} \qquad MTBF = MTTF + MTTR$$

MTTR : Mean time to repair
MTBF : Mean time between failures

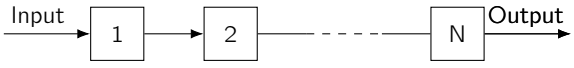
Failure in time

How many failures are to be expected?

Failure in time (FIT) is the number of failures that can be expected in 1×10^9 h of operation.

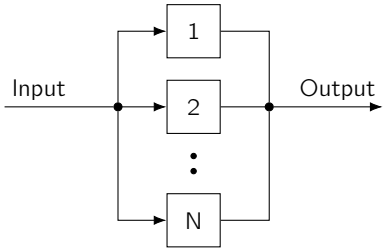
$$FIT = 1 \times 10^9 \cdot \frac{1}{MTBF}$$

Reliability modelling



$$R(t) = R_1(t) \cdot R_2(t) \cdot \cdots \cdot R_N(t) = \prod_{i=1}^{N} R_i(t)$$

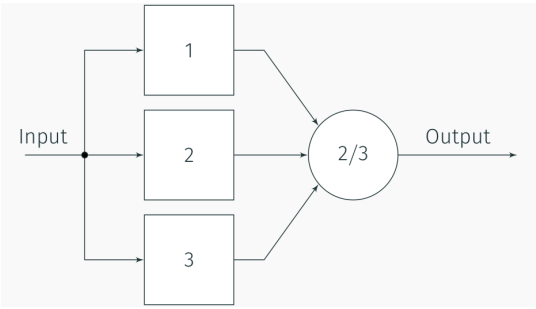
$$\lambda = \lambda_1 + \lambda_2 + \cdots = \sum_{i=1}^{N} \lambda_i$$



$$R(t) = 1 - Q(t) = 1 - \prod_{i=1}^{N}(1 - R_i(t))$$

$$Q(t) = Q_1(t) \cdot Q_2(t) \cdots Q_N(t) = \prod_{i=1}^{N} Q_i(t)$$

Triple Modular Redundancy

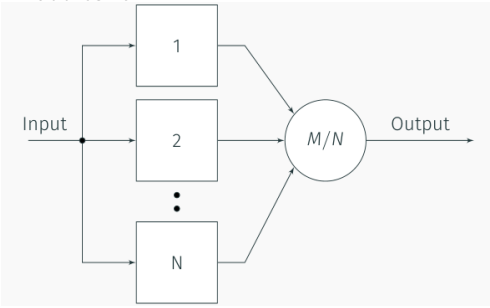


$$R_{TMR} = R_1(t) \cdot R_2(t) \cdot R_3(t) + (1 - R_1(t)) \cdot R_2(t) \cdot R_3(t) + (1 - R_2(t)) \cdot R_1(t) \cdot R_3(t) + (1 - R_3(t)) \cdot R_1(t) \cdot R_2(t)$$

For $R_1 = R_2 = R_3 = R_m$: $R(t) = 3 \cdot R_m^2(t) - 2 \cdot R_m^3(t)$

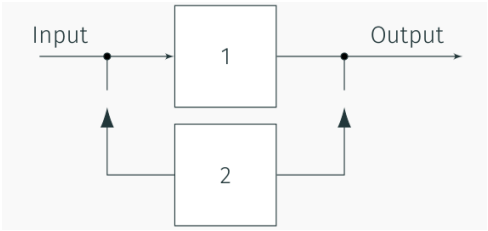
! N-Modular Redundancy

M out of N voting: System works correctly as long as **less then M** modules fail.



$$R_{MtoN}(t) = \sum_{i=0}^{N-M} \frac{N!}{(N-i)! \cdot i!} \cdot R_m^{N-i}(t) \cdot (1 - R_m(t))^i$$

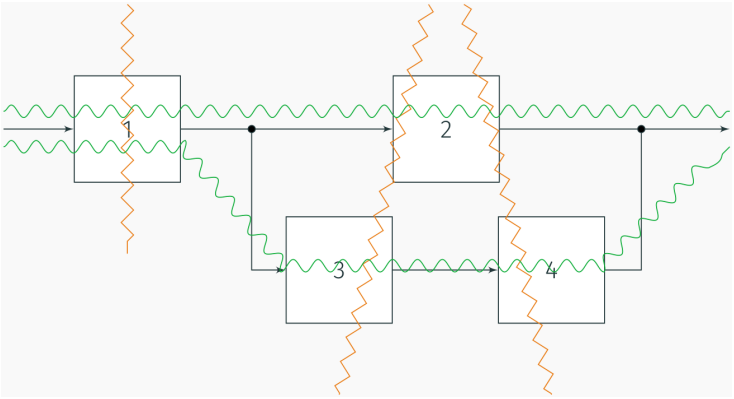
Dynamic Redundancy



$$R(t) = R_1(t) + (1 - R_1(t)) \cdot C_1 \cdot R_2(t) = R_m(t) + (1 - R_m(t)) \cdot C_m \cdot R_m(t)$$

R_n : Module Probabilities
C_n : Fault Coverage

Cut and Tie Sets



- ⇝ **cut** : sets of simultaneous failures leading to a system failure
- ⇝ **tie** : sets of working modules guaranteeing a working system

$$1 - \sum_{j=1}^{N_C} \prod_{i=1}^{n_j}(1 - R_i(t)) \leq R(t) \leq \sum_{j=1}^{N_T} \prod_{i=1}^{n_j}(1 - R_i(t))$$

Reliability prediction

Resistor (DoD MIL-Handbook 217)

$$\lambda_p = \lambda_b \cdot \pi_R \cdot \pi_Q \cdot \pi_E \quad [\text{failure}/1 \times 10^6 \text{h}]$$

λ_b : Base rate, temperature $(0.7 \times 10^{-3} \ldots 6.5 \times 10^{-3})$
π_R : Resistance range $(1.0 \ldots 2.5)$
π_Q : Quality of manufacturing $(0.03 \ldots 15)$
π_E : Environment $(1 \ldots 490)$
λ_p : Per part $(0.21 \times 10^{-3} \ldots 119.4)$

Capacitor (DoD MIL-Handbook 217)

$$\lambda_p = (C_1 \cdot \pi_T + C_2 \cdot \pi_E) \cdot \pi_Q \cdot \pi_L \quad [\text{failure}/1 \times 10^6 \text{h}]$$

C_1 : Die complexity
C_2 : Packaging
π_T : Ambient temperature
π_E : Environment
π_Q : Quality
π_L : Learning (production)

Prediction of software reliability

Task: estimate the number of faults within a given piece of software.

Prediction of software reliability

In general, this is a difficult task and still an active field of research.

- Assessment according to the techniques used during development and test.
- Rate the testing scheme used. Therefore, make minor changes (mutations) to the code and check their detection.
- Base upon experience from past projects.
- Estimate according to located faults located per time.

Number of faults $\propto$ unreliability?

It is *not* given that a program with more faults is less reliable than one with fewer faults!

Reliability assessment

Task: demonstrate that a system meets its reliability requirement.

How to...

... proof that a system fails less then once in 1×10^9 hour (i.e. $\approx$ 100000 year) of operation?
Trust the development techniques.

Software - Formal Methods

Examples

- B-Method – abstract machine notation, became Event-B, Rodin as tool
- ACSL – ANSI/ISO C specification language, Frama-C as tool
- Esterel – synchronous programming language, generates C code
- Z notation – specification language
- SPIN – model checker basing on Promela language
- SPARK – refinement of Ada

Spark

Guarantees

SPARK analysis can give strong *guarantees* that a program:

- does not read uninitialized data,
- accesses global data only as intended,
- does not contain run-time errors,
- respects key integrity properties,
- is a correct implementation of requirements.

Level of adoption

- Stone – valid SPARK subset of Ada
- Bronze – initialization and correct data flow
- Silver – absence of run-time errors (AoRTE)
- Gold – proof of key integrity properties
- Platinum – full functional proof of requirements

```
procedure Increment (X : in out Integer)
with
    Global => null,                          (1)
    Depends => (X => X),                     (2)
    Pre => X < Integer'Last,                 (3)
    Post => X = X'Old + 1                    (4)
  is
  begin
    X := X + 1;
  end Increment;
```

(1) **Global**: does not read or write any global variables
(2) **Dependance**: Value of **X** after call depends on the (previous) value of **X**
(3) **Condition**: `Increment` only callable if X $\leq$ max value
(4) **Conformance**: Check if the function does actually produce the desired result.

Vorsicht

This check is done by *Spark*, **NOT** *Ada*.
These properties are not only declared, but these are **proved** by a dedicated proof-engine `GNATprove`!

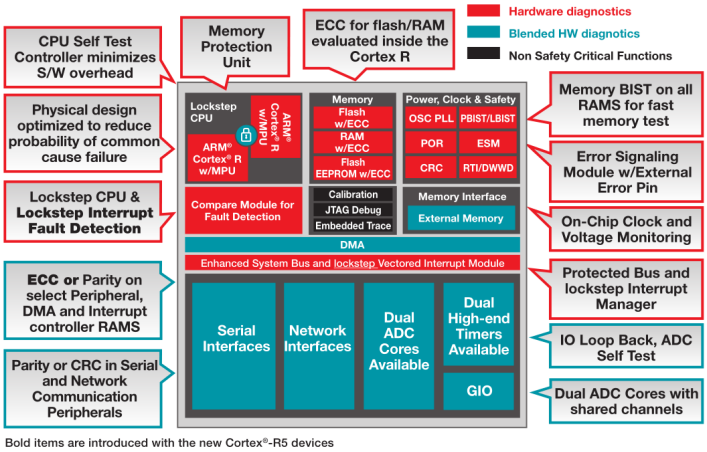
Hardware - Safety Processors

Hercules RM42 MCU

Safety features

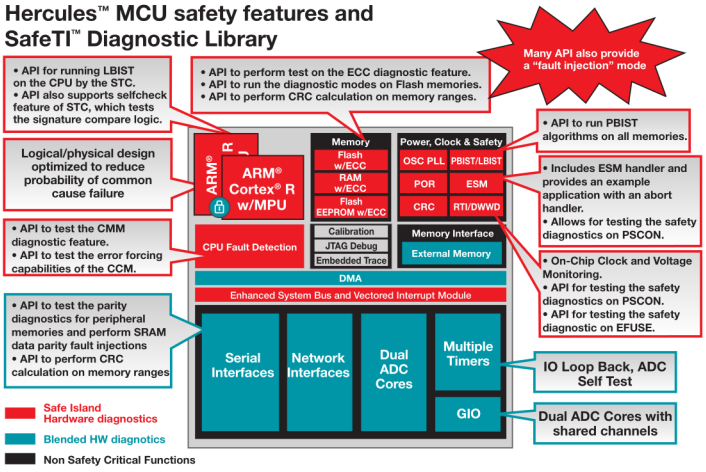
- dual ARM Cortex-R CPU
- cycle-to-cycle lockstep operation
- error correction code (EEC) circuit within CPU
- SRAM/Flash single bit error correction and double bit error detection (SECDED)
- CPU logic built-in self test (LBIST)
- SRAM programmable built-in self test (PBIST)
- lockstep interrupt manager
- advanced clock and voltage monitoring
- error signaling module with dedicated error pin

Hercules™ MCU safety features



Bold items are introduced with the new Cortex®-R5 devices

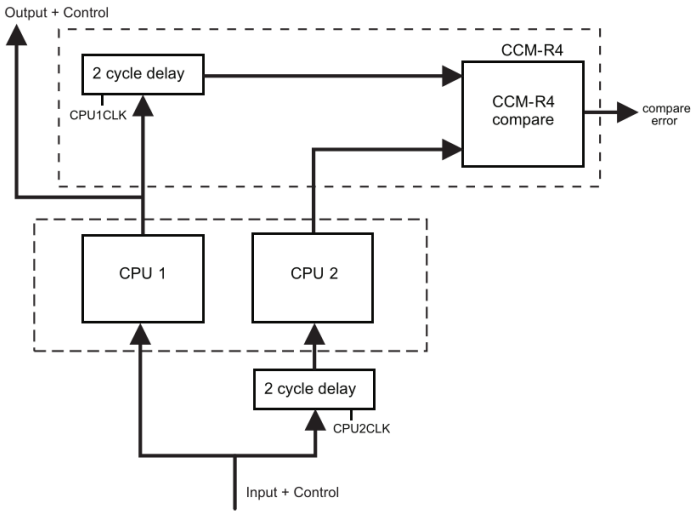
Hercules™ MCU safety features and SafeTI™ Diagnostic Library



Dual CPU

Lockstep operation

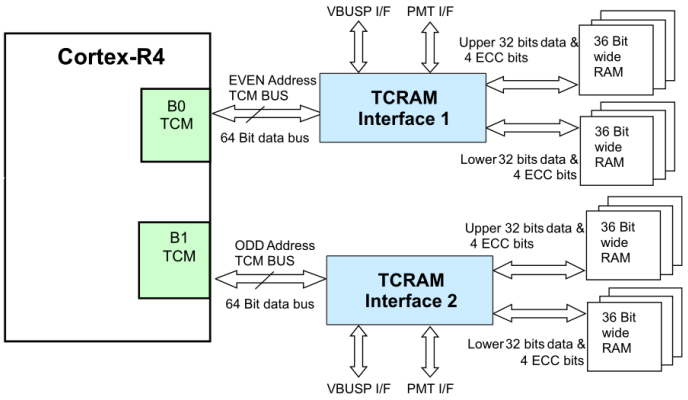
- two CPU, diverse placement ('north' & 'flip-west')
- operation shifted by two clock-cycles
- independent but synchronized clock sources



Memory

Tightly coupled RAM (TCRAM)

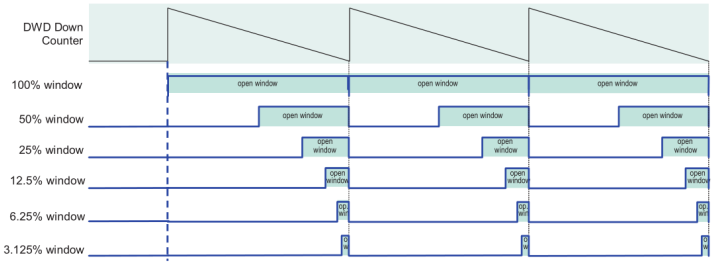
- 64-bit data and 8-bit ECC code (modified Hamming code)
- two 36-bit wide byte-interleaved RAM banks
- stores addresses for single-bit and multibit errors
- CPU address bus integrity checking
- redundant address decoding for chip select



Watchdog

Digital windowed watchdog (DWWD)

- configurable end time window
- reset by writing of correct sequence
- once enabled, it can not be disabled
- signalling via dedicated error-pin



Trends - Low Power

Firm-/Software Optimierungen

- `float` & `double` vermeiden → sehr Rechenintensiv bei Systemen ohne FPU (und auch sonst)
- Divisionen sind rechenintensiv, ausser durch Zweierpotenzen $2, 4, 8, 16, ...$, wo Bitshifting gemacht wird.
- Optimierungen einschalten beim Kompilieren :-)

Anhang

Crypto

Permutation Tabellen

Initial Permutation IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Final Permutation IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Bild von Spaghetti

I'm tired