

# Advanced Reliable Embedded Systems

ARES

Andi Ming / Quelldateien

## Table of contents

### Safety, Risiko Management

Terms . . . . .	1	2
Requirements . . . . .	2	2
Process (Iterative!) . . . . .	2	2
Verification, Validation & Certification (V&V&C) . . . . .	2	2
Hazard & Risk Analysis . . . . .	2	2
Risk Analysis . . . . .	2	2
Severity of Hazardous Event . . . . .	3	3
Frequency of Hazardous Event . . . . .	3	3
Risk Classification . . . . .	3	3
Integrity Classification . . . . .	3	3
Hardware Integrity . . . . .	3	3
Systematic Integrity . . . . .	3	3
Software Integrity . . . . .	3	3
Achieving Safety Integrity . . . . .	3	3
Core Design Process . . . . .	3	3
Fault Mitigation Strategies . . . . .	3	3
Fault Characteristics . . . . .	3	3
Key Challenges . . . . .	4	4
Fault Tolerance . . . . .	4	4
Reliability . . . . .	4	4
Unreliability . . . . .	4	4
Failure Rate . . . . .	4	4
Time-Variant Failure Rates . . . . .	5	5
Mean Times . . . . .	5	5
Mean Time to Failure . . . . .	5	5
Mean Time to Repair . . . . .	5	5
Mean Time Between Failures . . . . .	5	5
Failure in Time . . . . .	5	5
Reliability Modelling . . . . .	5	5
Series Systems . . . . .	5	5
Parallel Systems . . . . .	5	5
Redundancy . . . . .	5	5
Software Safety . . . . .	6	6
Capability Maturity Model (CMM) . . . . .	6	6
Formal Methods . . . . .	6	6
Frama-C . . . . .	6	6
Evolved Value Analysis (EVA) . . . . .	6	6
Weakest Precondition (WP) . . . . .	6	6
Encryption . . . . .	7	7
Cesar Cipher / Substitution Cipher . . . . .	7	7
Enigma Maschine  . . . . .	7	7
Stream & Block Cipher . . . . .	7	7
Konfusion . . . . .	7	7
Diffusion . . . . .	7	7
Feistel Network  . . . . .	7	7
XTEA . . . . .	7	7
Advanced Encryption Standard (AES) . . . . .	8	8
Key Scheduler / Subkey Generierung . . . . .	9	9
Entschlüsselung . . . . .	10	10
Cipher Modi  . . . . .	11	11
Electronic Code Book (ECB) . . . . .	11	11
Cipher Block Chaining (CBC) . . . . .	11	11
Output Feedback (OFB) . . . . .	11	11
Cipher Feedback (CFB) . . . . .	11	11
Counter (CTR) . . . . .	11	11
Galois Counter Mode (GCM) . . . . .	12	12
Double Encryption & Meet-In-The-Middle Attack (MITM) . . . . .	12	12
Triple Encryption . . . . .	12	12
Key Whitening . . . . .	12	12
Asymmetrische Kryptographie . . . . .	12	12
Key Distribution Problem . . . . .	12	12
Public Key Cryptography  . . . . .	13	13
Diffie Hellman  1,  2 . . . . .	13	13
Elgamal Encryption Scheme . . . . .	14	14
RSA . . . . .	14	14
Elliptic-Curve-Cryptography (ECC) . . . . .	14	14
Elliptic-Curve Diffie-Hellmann Key Exchange (ECDH) . . . . .	15	15
Security Services . . . . .	15	15
RSA Signature Scheme . . . . .	15	15
Digital Signature Algorithm (DSA) . . . . .	16	16
Elliptic Curve Digital Signature Algorithm (EC-DSA) . . . . .	16	16
Key Freshness . . . . .	16	16
Key Distribution Center (KDC) . . . . .	16	16
Asymmetric Key Distribution (DHKE, RSA, ECDSA...) . . . . .	16	16
Certifying Authority (CA) . . . . .	16	16
Hash Functions . . . . .	17	17
SHA-1 . . . . .	17	17
SHA-3 (Keccak) . . . . .	17	17
weitere Begriffe . . . . .	17	17

### Reliability

17

### Vulnerability

17

### Boot

17

### Memory

17

### Execution

17

### Fault Handling

17

## Safety, Risiko Management

Safety | Safety-Critical

**Safety** is defined as preventing harm to humans/environment, while **safety-critical systems** ensure this property.

### Encryption

7

Cesar Cipher / Substitution Cipher . . . . .	7
Enigma Maschine  . . . . .	7
Stream & Block Cipher . . . . .	7
Konfusion . . . . .	7

## ! Risk | Safety-Integrity

**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.

## Terms

- **Hazard**: A situation in which there is actual or potential danger for people or environment.
- **Accident**: Unintended event harming people or environment.
- **Incident**: Unintended event which does not harm, but has the potential to do so.
- **Risk**: Likelihood of hazard occurrence, and the likely consequences. Risk = Severity × Probability
- **Fault**: Defect in system. Can be **random** or **systematic**.
- **Error**: Deviation from the required operation of the system.
- **System Failure**: Occurs when system fails to perform its required function.
- **Casualties** (Kausalitäten): The presence of a fault *may* lead to an error, which *may* lead to a system failure, which *may* lead to an accident.

## Requirements

Requirements give a system the properties of **integrity** and **dependability**.

This demands: (1) **Safety**, (2) **Reliability**, (3) **Availability**, (4) **Maintainability**.

### ⚠ Conflicts

In general, the various requirements to a system are conflicting among themselves.

## Process (Iterative!)

1. Identification of hazards associated with the system
2. Classification the hazards
3. Determination of methods to deal with hazards
4. Assignment of reliability and availability requirements
5. Determination of safety integrity level
6. Specification of development method appropriate to integrity level

## Verification, Validation & Certification (V&V&C)

- **Verification**: Confirms system meets specifications
- **Validation**: Ensures fitness for intended purpose
- **Certification**: Obtains regulatory approval through evidence documentation
- Key distinction example: Medical device passing lab tests (verification) but failing clinical trials (validation)

## Hazard & Risk Analysis

### • Hazard identification methods:

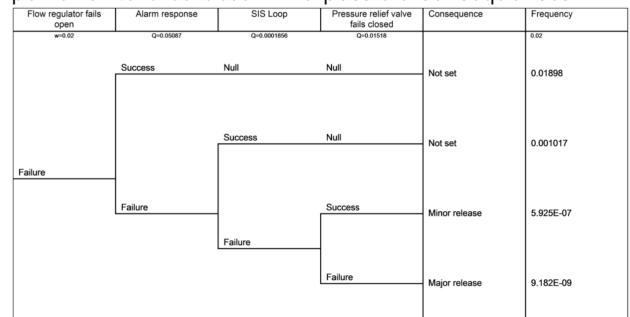
- **FMEA (Failure mode and effects analysis)**: Analyzes component failure effects on ultimate consequences.

Failure Mode Effects Analysis												
System Description: Landing Gear Operation Mode: Flight - Level 2												
Item Number	Item Description	Function	Fun. Id.	Failure Mode	Local Effect	Next Higher Effects	End Effects	Sev.	Detection Method	Compensation Provisions	Remarks	
1.1.1	Main Pump	Provides pressure when requested by Pilot Command	1	Fails to operate	No effect during this phase	No effect during this phase	No effect	IV	Indication to pilot	None		
				Untimely operation	Untimely hydraulic generation from Main Hydraulic Generation Assembly	Untimely hydraulic generation from Main Hydraulic Generation Assembly to Actuator Assembly	Untimely extension of Landing Gear	I	Indication to pilot	None		
1.1.2	Check Valve (Main)	Prevents reverse flow	1	Stucked closed	Loss of fluid flow through the Main Hydraulic Generation Assembly check valve	No effect during this phase	No effect	IV	Indication to pilot	None		
				Stucked open	Permits fluid flow through the main hydraulic valve when not required	No effect during this phase	No effect	IV	Undetected	None		

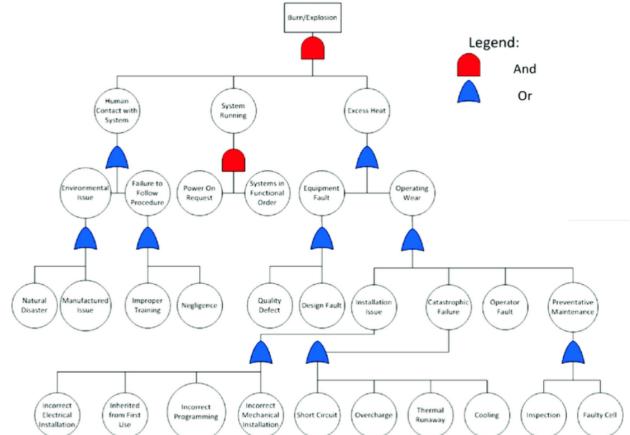
- **HAZOP (Hazard and operability studies)**: Uses guide-words to detect operational deviations.

Guide Word	Deviation	Causes	Consequences	Action
NO	No cooling	Cooling water valve malfunction	Temperature increase in reactor	Install high temperature reactor 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 ineffective and effect on the reaction	Less cooling, TAH will detect. If detected, isolate water source. Back up water source?

- **ETA (Event tree analysis)**: Model effects from starting point forward to determine possible consequences.



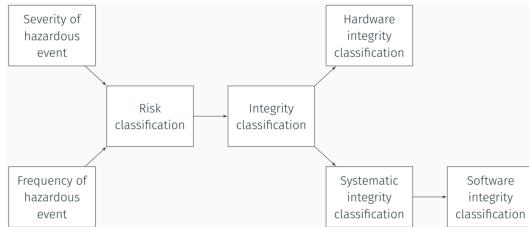
- **FTA (Fault tree analysis)**: Identify hazards and determine their possible causes.



## Risk Analysis

- **Risk classification** combines severity (catastrophic/negligible) and frequency (frequent/incredible). Risks are categorized as intolerable (I) to negligible (IV).

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



Safety Integrity Level	Continuous mode (prob. of dangerous failure per year)	Demand mode (prob. of failure to perform on demand)
4	$\geq 1 \times 10^{-5}$ to $1 \times 10^{-4}$	$\geq 1 \times 10^{-5}$ to $1 \times 10^{-4}$
3	$\geq 1 \times 10^{-4}$ to $1 \times 10^{-3}$	$\geq 1 \times 10^{-4}$ to $1 \times 10^{-3}$
2	$\geq 1 \times 10^{-3}$ to $1 \times 10^{-2}$	$\geq 1 \times 10^{-3}$ to $1 \times 10^{-2}$
1	$\geq 1 \times 10^{-2}$ to $1 \times 10^{-1}$	$\geq 1 \times 10^{-2}$ to $1 \times 10^{-1}$

## Severity of Hazardous Event

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 of Hazardous Event

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

## Risk Classification

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 Classification

**ALARP-Rule:** Class II & III is only acceptable if it is **As Low As Reasonably Practicable**

Risk can be reduced by safety features. Achieved reduction depends upon integrity of these features.

Safety integrity is how likely a safety system is to perform its job correctly, under all conditions, and for the required time.

## Safety Integrity Levels (SIL)

## Hardware Integrity

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

## Systematic Integrity

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

## Software Integrity

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

## Achieving Safety Integrity

The process involves iterative design stages and layered fault mitigation strategies to meet safety-critical system requirements.

## Core Design Process

- Abstraction:** Identify essential system properties
- Decomposition:** Break systems into analyzable components
- Elaboration:** Add implementation details
- Decision:** Select optimal design alternatives

## Fault Mitigation Strategies

Four complementary approaches:

- Avoidance:** Prevent faults during design phase
- Removal:** Eliminate faults through testing/reviews
- Detection:** Identify faults during operation
- Tolerance:** Maintain functionality despite faults

## Fault Characteristics

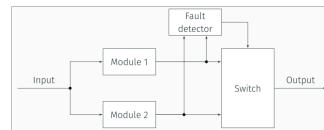
Category	Types	Examples
Nature	Random (HW) vs Systematic	$\alpha$ -particle errors
Duration	Permanent/Transient/Intermittent	Broken chip vs
Extent	Localized vs Global	Single sensor vs

### Hardware Fault Tolerance:

- Static** (TMR/NMR): Mask faults via majority voting (3-5 modules)
- Dynamic**: Detect & switch to backups
- Hybrid**: Combine masking + reconfiguration

### Software Fault Tolerance:

- **N-version Programming:** Parallel diverse implementations (Airbus/Shuttle)
- **Recovery Blocks:** Fallback modules with acceptance tests
- **Information:** Additional data (parity / checksum)
- **Temporal:** Repeat calculations



- + allows one module 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

## Key Challenges

- Common-mode failures require **diversity** in:
  - Implementation methods
  - Programming languages
  - Hardware platforms
- **Systematic faults (spec/design errors)** are harder to mitigate than random HW faults
- No single technique provides complete protection

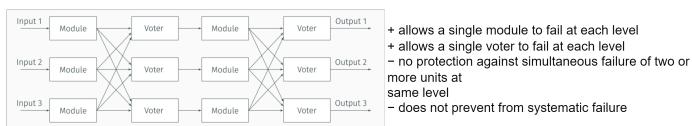
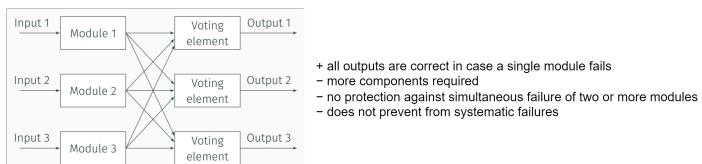
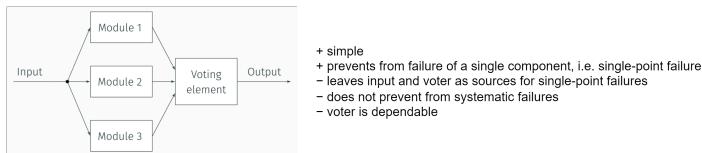
**Critical Insight:** Achieving safety integrity requires combining multiple fault mitigation strategies through iterative design refinement, as perfect fault elimination is impossible in complex systems.

**Detection:** Functional checking, Consistency checking, Signal comparison, Checking pairs, Information redundancy, Instruction monitoring, Loopback testing, Watchdog timers, Bus monitoring, Power supply monitoring

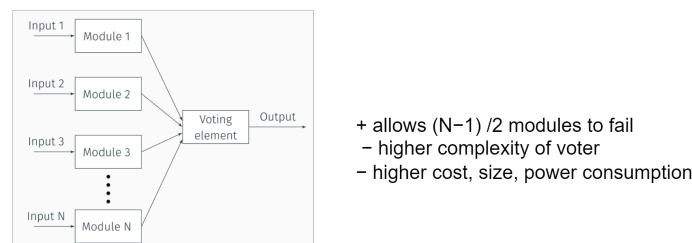
## Fault Tolerance

### • Redundancy strategies:

- **TMR (Triple Modular Redundancy):** Voting systems mask faults via majority logic (3 modules)

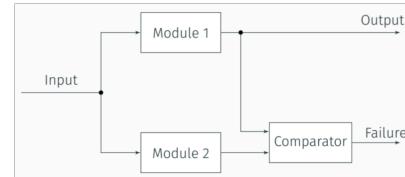


- **NMR:** Voting systems mask faults via majority logic (3-5 modules). Allows for  $\frac{N-1}{2}$  modules to fail



- **Dynamic redundancy:** Switches to backup modules after fault detection.

- **Self checking pair:** The outputs are compared and give indication of failure



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

- **Diversity:** Combines different implementations/languages to avoid common-mode failures.
- **Software fault tolerance:** Uses *N-version programming* (parallel implementations) or *recovery blocks* (fallback modules with acceptance tests).

## Reliability

### i Reliability

Reliability  $R$  is the probability of a component or system functioning correctly over time  $R(t)$ . Describing a statistical behaviour of a component or system.

Given: Period of time, set of operating conditions.

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

with  $n(t)$  number of working elements, and  $N$  number of original elements.

### ⚠ According to DoD MIL-Handbook

The reliability prediction according to the **MIL-Handbook** (US) is not followed and maintained anymore because of too many unknown variables.

## Unreliability

Probability  $Q(t)$  that a system will **not** function over a given period of time.  $Q(t) + R(t) = 1$

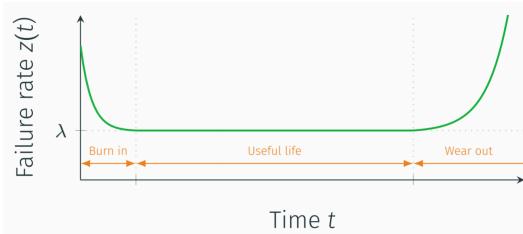
$$Q(t) = \frac{n_f(t)}{N} = 1 - R(t)$$

with  $n_f(t)$  number of failed components at time  $t$ .

## Failure Rate

The rate  $z(t)$  at which a device fails

$$z(t) = \frac{1}{n(t)} \cdot \underbrace{\frac{dn_f(t)}{dt}}_{\text{Failures}}$$



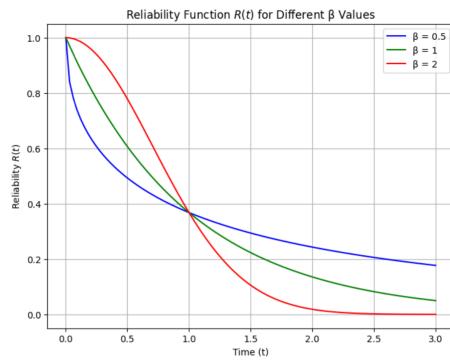
For a constant **failure rate**  $z(t) = \lambda$  the probability of a system working correctly decreases exponentially

$$R(t) = e^{-\lambda t}$$

## Time-Variant Failure Rates

Software failures which are systematic and correctable the failure rate decreases with time. **Weibull** distribution

$$R(t) = e^{-(\frac{t}{\eta})^\beta}$$



## Mean Times

### Mean Time to Failure

Expected time before first failure

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

### Reliability

With  $\lambda = 0.001$  failure/h  $MTTF = 1000h$ .  
But at  $t = 1000h$  the reliability is only  $R(t) \approx 0.37$  (chance for running at 1000h mark is 37%)

### Mean Time to Repair

Time to repair given by repairability  $\mu$

$$MTTR = \frac{1}{\mu}$$

### Mean Time Between Failures

$$MTBF = MTTF + MTTR$$

## Failure in Time

Number of failures expected in  $1 \times 10^9 h$  of cumulative operation hours

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

## Reliability Modelling

### Series Systems

Failure of **any component fails**

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

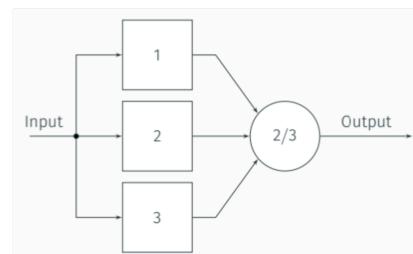
$$\lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N = \sum_{i=1}^N \lambda_i$$

### Parallel Systems

System operational as long as **one component is functioning**

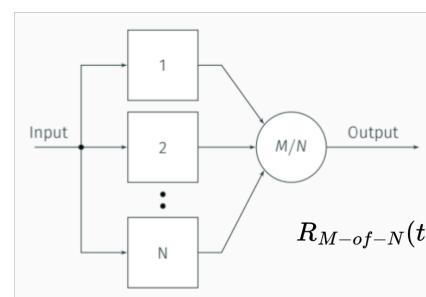
$$R(t) = 1 - Q(t) = 1 - \prod_{i=1}^N (1 - R_i(t))$$

### Redundancy

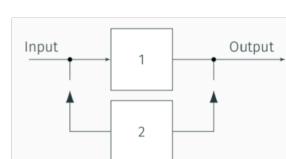


$$R_1 = R_2 = R_3 = R_m$$

$$R(t) = 3R_m^2(t) - 2R_m^3(t)$$



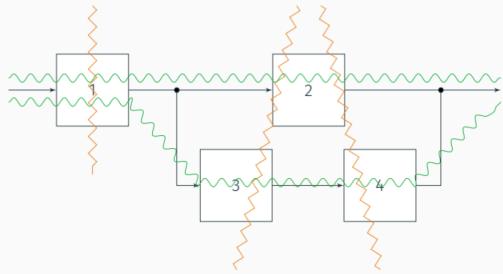
$$R_{M-of-N}(t) = \sum_{i=0}^{N-M} \frac{N!}{(N-i)!i!} \cdot R_m^{N-i}$$



If module 1 fails,  
module 2 is activated

Fault Coverage  $C_m$

$$R(t) = R_m(t) + (1 - R_m(t))C_mR_m(t)$$



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

## Boundaries

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

### ⚠ Reliability Prediction

There is extensive (usually MIL std.) literature but often with lots of unknown variables.

### ❗ Reliability Assessment

How to proof that a system fails less than once in  $1 \times 10^9$  hour (i.e.  $\approx 100\,000$  year) of operation?  
 Trust the development techniques.

## Software Safety

Common faults:

Coding faults, logical errors within calculations, numeric under- and overflows, stack under- and overflows, range under- and overflows (arrays!), uninitialised variables, unintended side effects, truncation by casts, rounding effects, memory leaks, ...

## Capability Maturity Model (CMM)

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

LOC: Lines of Code

## Formal Methods

Apply mathematically rigorous techniques for the specification development and verification of the software and hardware systems.

### Examples

- B-Method – abstract machine notation, became Event-B, Rodin as tool
- Esterel – synchronous programming language, generates C code
- Z notation – specification language
- SPIN – model checker basing on Promela language
- SPARK – refinement of Ada also possible to have a program in Ada and submodules in spark (more save)
- Frama-C – basing on ACSL specification language,

## Frama-C

- Frama-C is an open source framework
- core to read C files and build abstract syntax trees
- set of plug-ins to do static analysis and to annotate syntax trees
- plug-ins can collaborate, i.e. use another plug-in
- plug-ins programmed in OCaml language
- major plug-ins: EVA & WP
- ACSL (ANSI/ISO C Specification Language) for annotations by C comments /\* @ ... \*/

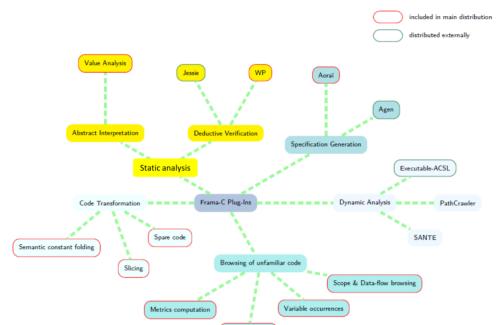


Figure 0.1: Frama-C plug-ins

## Evolved Value Analysis (EVA)

Computes variation domains for variables, determine bounds for the values of variables and function results.

```
int abs(int x) {
    if (x < 0)
        return -x;
    else
        return x;
}
```

pipistrellus@N0007494:~\$ frama-c -eva example\_1.c -main abs
[Kernel] Parsing example\_1.c (with preprocessing)
[eva] Analyzing a complete application starting at abs
[eva] initial-state: Values of globals at initialization
[eva] alarm] example\_1.c:3: Warning: signed overflow. assert -x ≤ 2147483647;
[eva] ===== VALUES COMPUTED =====
[eva] final-state] Values at end of function abs:
...retres ∈ [0..2147483647]

## Weakest Precondition (WP)

Proofing certain properties, tries to proof the properties defined.

```
/*@ ensures \result == (a+b)/2;
@ assigns \nothing;
@ */
int mean(int a, int b) {
    return (a+b)/2;
}
```

pipistrellus@N0007494:~\$ frama-c -wp example\_4.c
[Kernel] Parsing example\_4.c (with preprocessing)
[wp] Warning: Missing RTE guards
[wp] 2 goals scheduled
[wp] Proved goals: 4 / 4
Terminating: 1
Unreachable: 1
Qed: 2

### ℹ Missing RTE guards

In this example the specification is met as long as *no runtime error* is issued (see warning Missing RTE guards). Add the flag `-wp-rte` to add additional conditions.  
 It can not be guaranteed that no overflow could occur.

### ❗ High integrity software

High integrity software is possible, but this demands a lot of **efforts** and **passion** from it's developers.

## Encryption

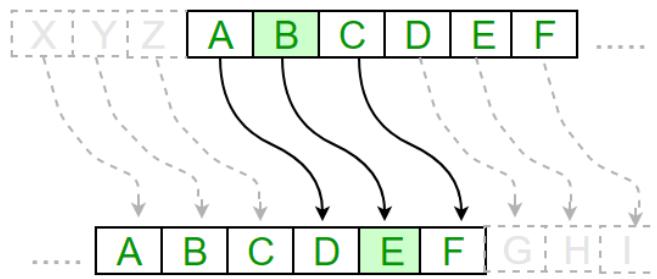
### Note

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 aufgefüllt werden.

### ! Sicherheit beweisen

Sicherheit kann nicht bewiesen werden, unsicherheit schon!

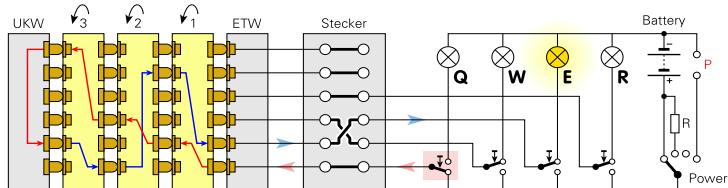
## Cesar Cipher / Substitution Cipher



Buchstaben werden um  $x$  Positionen verschoben (z.B.  $A \rightarrow C$ ). Nachteil ist, dass die Entschlüsselung sehr einfach ist.

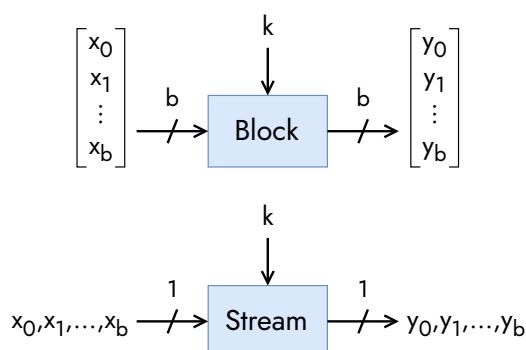
## Enigma Maschine

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



Nach jedem 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.

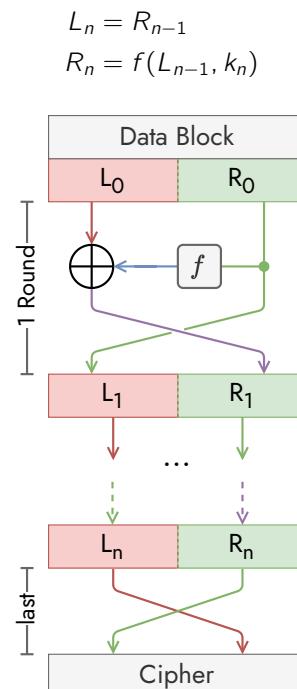
Konfusion erhöht die Mehrdeutigkeit des Ciphertextes und wird sowohl von Block- als auch von Stream-Ciphern verwendet.

## 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 →  $2 \times 32$ -Bit). Eine Runde entspricht:



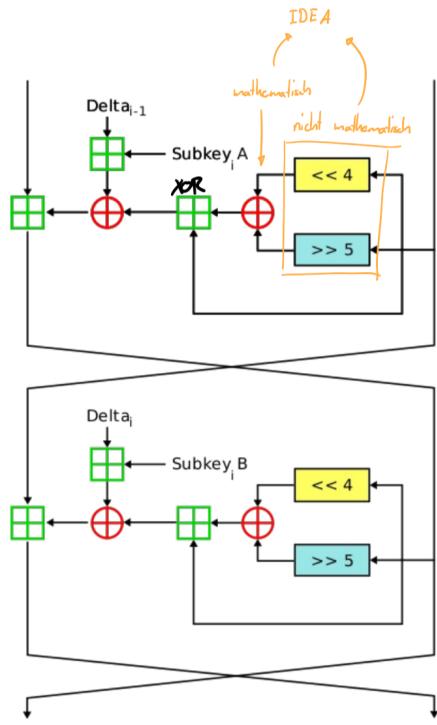
Funktion  $f$  ist **wichtig**. Wenn diese sicher gegen Attacken ist, dann wird das Feistel Netzwerk mit jeder Runde und Key-Segment sicherer!

### ! Ver- & Entschlüsseln

Verschlüsselte Informationen können mit dem genau gleichen Ablauf wieder entschlüsselt werden.

## XTEA

eXtended TEA ist eine Erweiterung von TEA, welcher die Verschlüsselung besser macht. Gleiche Eigenschaften + Konstantwert delta=0x9E3779B9.



Combination of mathematical (*addition*) and non-mathematical (*shift*) operations comes from *IDEA* (developed at ETH).

```
void encipher (unsigned int num_cycles, uint32_t
  v[2], uint32_t const k[4]) {
    unsigned int i;
    const uint32_t delta = 0x9E3779B9; //decipher
    changes:
    uint32_t v0 = v[0], v1 = v[1], sum = 0; //, sum =
    delta * num_cycles;
    for (i=0; i < num_cycles; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum +
        k[sum & 3]); //-=, exchange lines above and
        sum += delta; //-=, exchange lines above and
        below
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum +
        k[(sum>>11) & 3]); //-
    }
    v[0] = v0; v[1] = v1;
}
```

### ! 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!

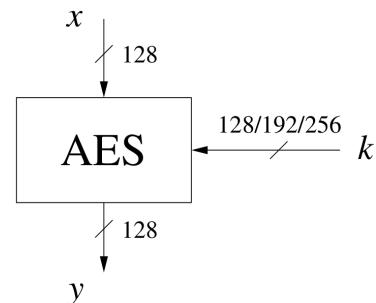
### i Anforderungen & Kandidaten

#### Anforderungen

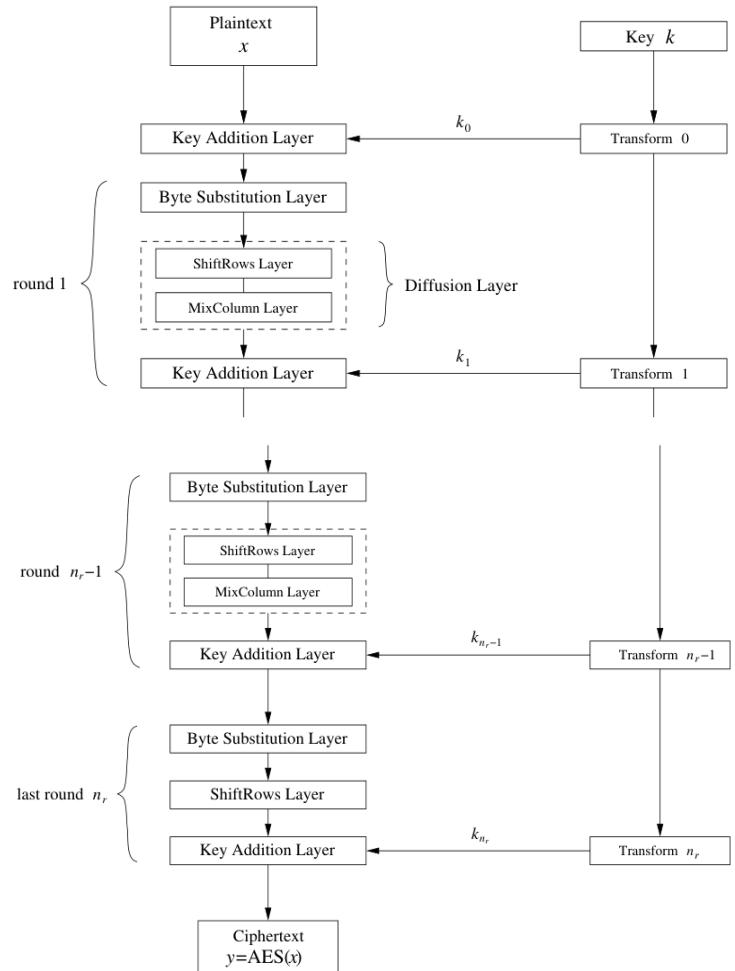
- block cipher with 128 bit block size (16-bytes)
- three key lengths: 128, 192 and 256 bit
- security relative to other submitted algorithms
- efficiency in software and hardware

#### 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



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



**Key Addition Layer** Ein 128b Round-Key/Subkey (vom Hauptkey) wird in das Datenpaket gexored (Key Whitening)

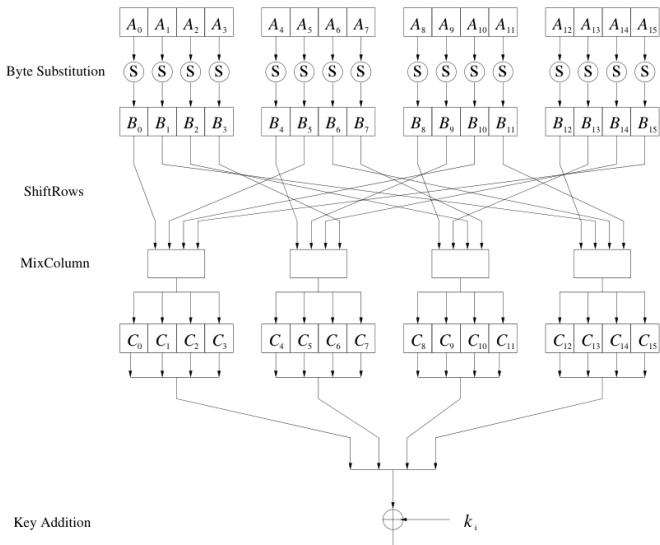
**Byte Substitution layer (S-Box)** Eine Nichtlineare Datentransformation (Konfusion)

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

### ⚠ Warning

Letzte Runde macht **keinen** MixColumn!

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



Für die Byte Substitution wird folgende S-Box verwendet:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	y
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	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	F8	A1	89	09	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	

**nonlinear** (for mathematical confusion):  $f(A) + f(B) \neq f(A + B)$

Shiftrows basiert auf folgendem Verschiebungsmuster.

Input	Output
$B_0 \ B_4 \ B_8 \ B_{12}$	$B_0 \ B_4 \ B_8 \ B_{12}$ no shift
$B_1 \ B_5 \ B_9 \ B_{13}$	$B_5 \ B_9 \ B_{13} \ B_1$ ← 1 position left shift
$B_2 \ B_6 \ B_{10} \ B_{14}$	$B_{10} \ B_{14} \ B_2 \ B_6$ ← 2 positions left shift
$B_3 \ B_7 \ B_{11} \ B_{15}$	$B_{15} \ B_3 \ B_7 \ B_{11}$ ← 3 positions left shift

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

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

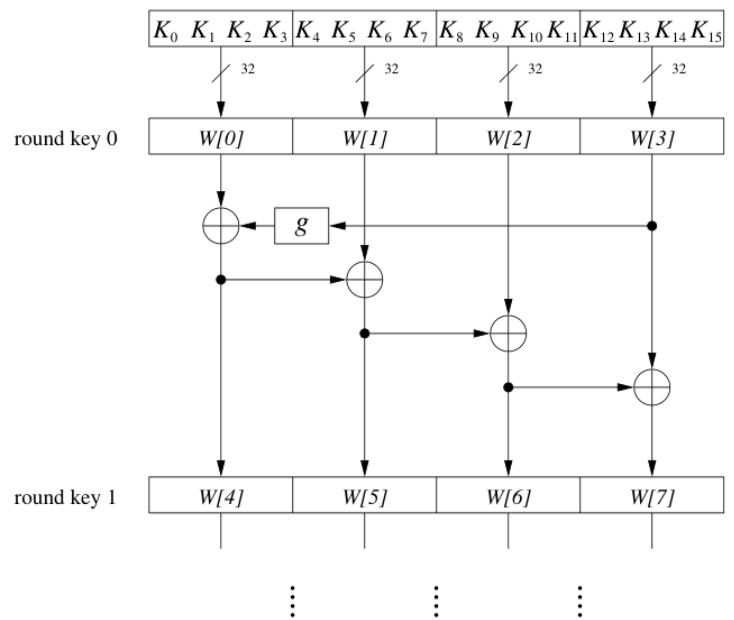
01, 02, 03 sind Representationen eines Elements von  $GF(2^8)$ .

**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  $25_h, 25_h, 25_h, 25_h$

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

### Key Scheduler / Subkey Generierung

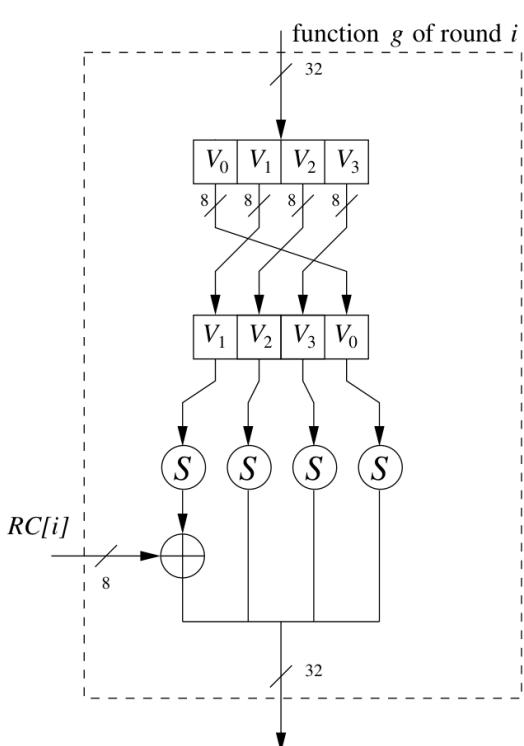


Wie in der Abbildung zu sehen ist, wird das ganz linke Word eines Subkeys  $W[4i]$ , wobei  $i = 1, \dots, 10$  ist, berechnet als:

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

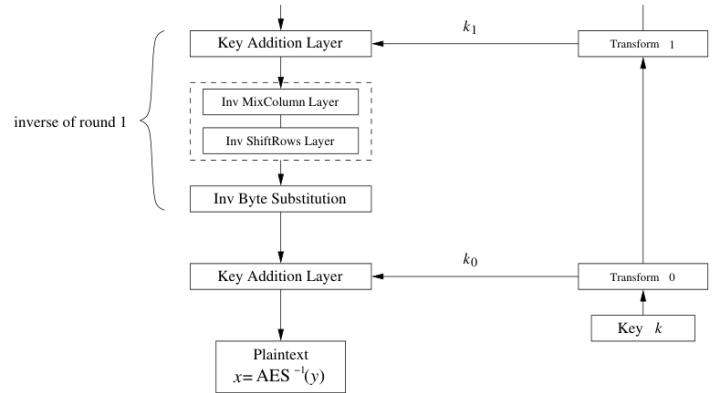
### g-Funktion

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.



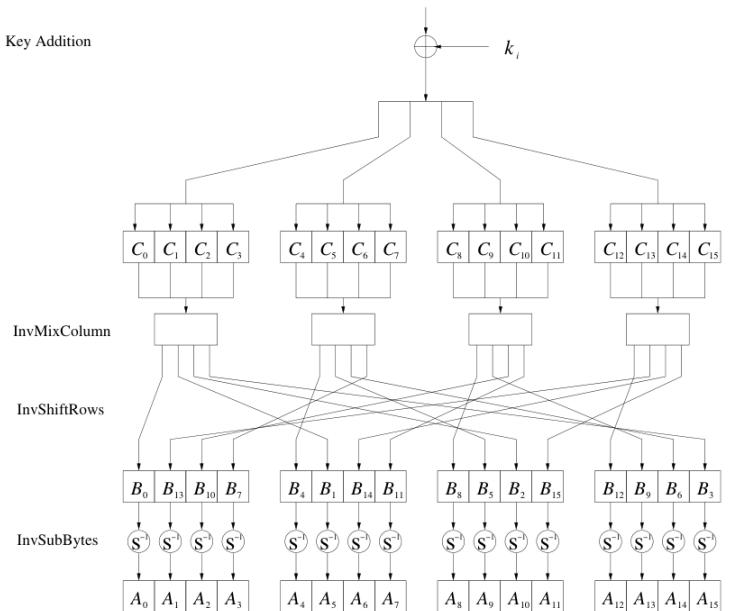
Die  $g$ -Funktion rotiert die vier input bytes, führt eine byteweise S-Box substitution und fügt einen Runden-Koeffizient  $RC$  dazu. Der Runden-Koeffizient ist ein Element vom Galois field  $GF(2^8)$  und wird nur am Byte ganz links dazugefügt. Die Koeffizienten variieren von Runde zu Runde mit folgender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

$$RC[ ] = [(0 \times 01)_{16}, (0 \times 02)_{16}, (0 \times 04)_{16}, (0 \times 08)_{16}, (0 \times 10)_{16}, (0 \times 20)_{16}, (0 \times 40)_{16}, (0 \times 80)_{16}, (0 \times 1B)_{16}, (0 \times 36)_{16}]$$



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

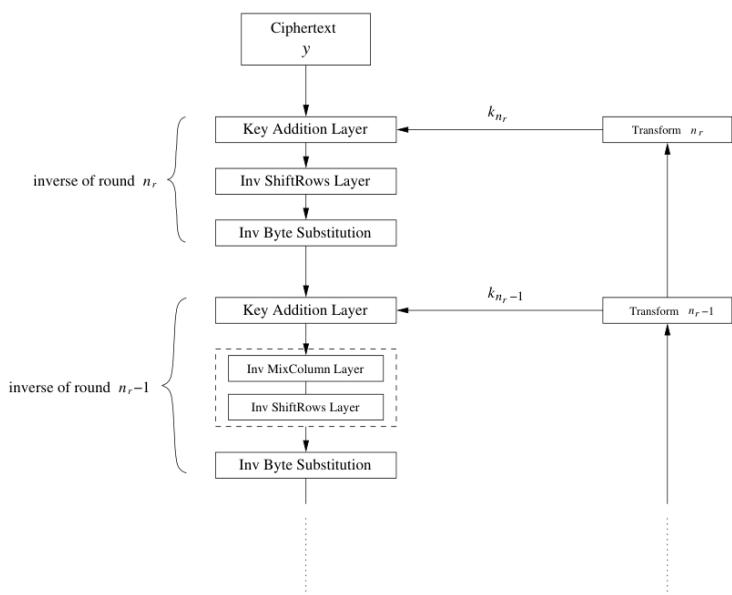
**Round funktion** Die Sublayers werden ebenfalls verkehrt abgelaufen! Die Inverse der MixColumn Sublayer verläuft auf dem gleichen GF-Prinzip, wiederum einfacher umgekehrt.



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

## 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  $k_9, \dots$



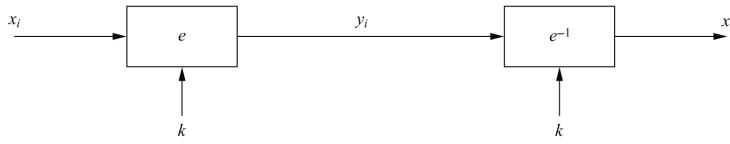
Input	Output
$B_0 \ B_4 \ B_8 \ B_{12}$	$B_0 \ B_4 \ B_8 \ B_{12}$ no shift
$B_1 \ B_5 \ B_9 \ B_{13}$	$B_{13} \ B_1 \ B_5 \ B_9$ → 1 position right shift
$B_2 \ B_6 \ B_{10} \ B_{14}$	$B_{10} \ B_{14} \ B_2 \ B_6$ → 2 positions right shift
$B_3 \ B_7 \ B_{11} \ B_{15}$	$B_7 \ B_{11} \ B_{15} \ B_3$ → 3 positions right shift

$x$	0	1	2	3	4	5	6	7	$y$	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	
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 Modus W

### 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**

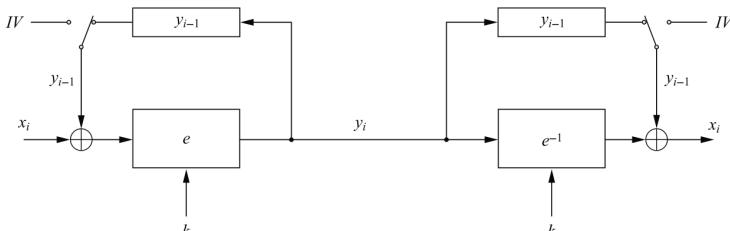
### Substitution Attack on ECB

The attacker sits in the middle in between bank A and bank B and controls all traffic:  
**"Man in the middle attack"** this is the typical scenario in crypto world

- The attacker sends \$1.00 transfers from his account at bank A to his account at bank B repeatedly
- He can check for ciphertext blocks that repeat, and he stores blocks 1,3 and 4 of these transfers
- He now simply replaces block 4 of other transfers with the block 4 that he stored before
- All transfers from some account of bank A to some account of bank B are redirected to go into the attacker's B account
- Works completely without attacking the block cipher itself!

Block #	1	2	3	4	5
Sending Bank A	Sending Account #	Receiving Bank B	Receiving Account #	Amount \$	

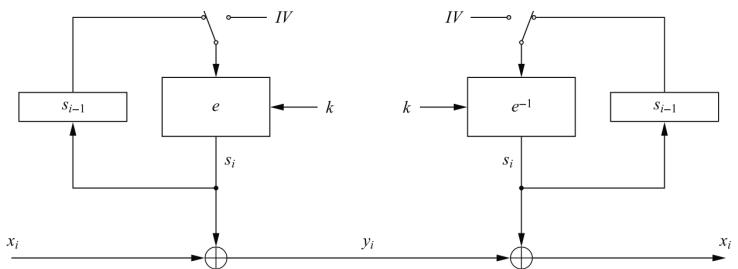
### Cipher Block Chaining (CBC)



$X_1$  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

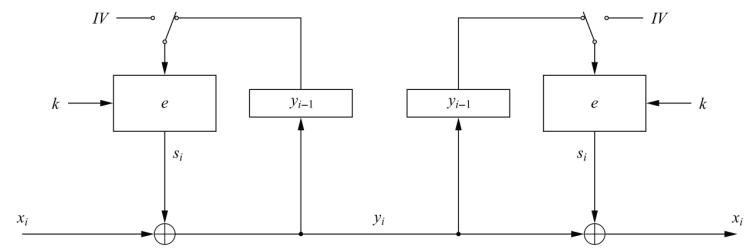
### Output Feedback (OFB)



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)

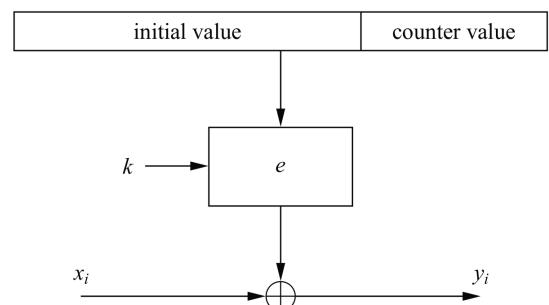


Block Cipher zu **asynchronem** Stream Cipher (**asynchron**: Key stream  $S_i$  ist vom Cipher abhängig und ist daher eher langsamer im Vergleich zu OFB).

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

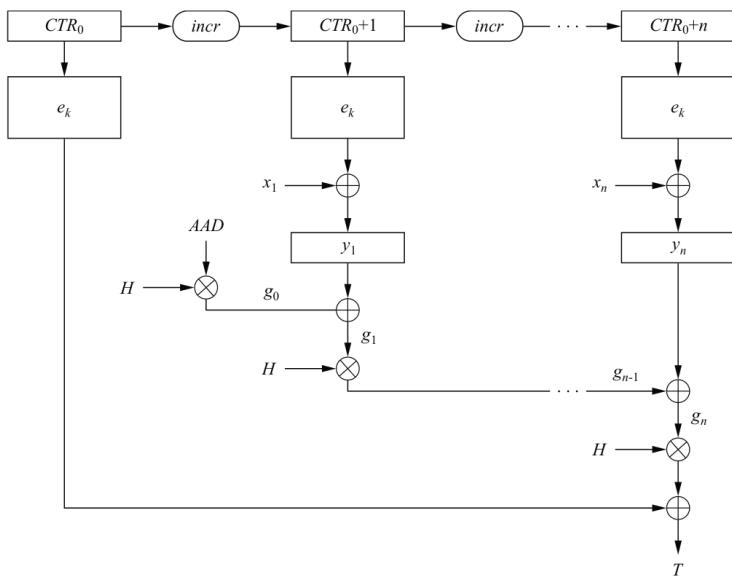
### Counter (CTR)

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 cipher)
- + Very fast, can be precomputed
- Ciphertext  $y_i$  only depends on plaintext  $x_i$ ,  $k$

## Galois Counter Mode (GCM)



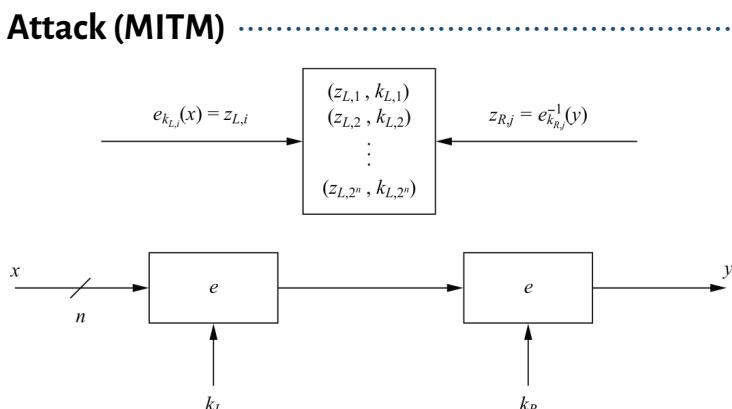
Fügt **message authentication & integrity** als CTR-Erweiterung ein. Alle Multiplikationen werden mit dem 128-Bit Galois Feld  $GF(2^{128})$  und dem irreduziblen Polynom  $P(x) = x^{128} + x^7 + x^2 + x + 1$  gemacht.

+ Authentication data can be left in plain text

**i** 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 Encryption & Meet-In-The-Middle Attack (MITM)



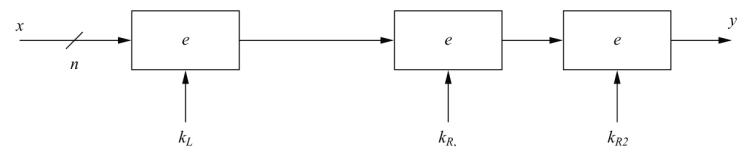
**Double Encryption** wird der Plaintext  $x$  zuerst mit  $k_L$  verschlüsselt und dann mit  $k_R$  folgend entschlüsselt  $\rightarrow$  Ciphertext  $y$ .

**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 Komplexität von  $2^{2k}$  zu  $2^{k+1}$  Suchoperationen.

## Triple Encryption

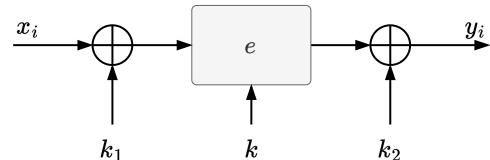


Der Plaintext wird *encrypted-decrypted-encrypted* (EDE) mit drei verschiedenen Keys. Bei einer Key-Größe von 56-Bits (168-Bit Key komplett) verlängert sich der Aufwand zurück zu  $2^{2k}$  Operationen.

$$y = e_{k_1}(e_{k_2}^{-1}(e_{k_3}(x)))$$

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.

### ⚠ Warning

Schützt den Blockcipher nicht vor **analytischen** Angriffen wie lineare oder differentiale Kryptoanalyse. Daher ist dieser nutzlos bei schwachen Cipher (z.B. DES).

## Asymmetrische Kryptographie

**i** Key Lengths and Security

Symmetric	ECC	RSA, DL	Remark
64 Bit	128 Bit	$\approx 700$ Bit	Only short term security (a few hours or days)
80 Bit	160 Bit	$\approx 1024$ Bit	Medium security (except NSA)
128 Bit	256 Bit	$\approx 3072$ Bit	Long term security (without quantum computers*)

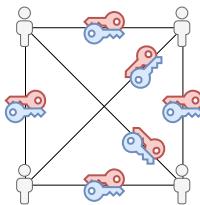
Paar & Pelzl: Quantum Computing is "at least 2-3 decades away, and some people doubt that QC will ever exist"

- **Euclidean Algorithm** (find inverse d): complexity grows linearly with the number of bits
- **Euler's Phi Function**: How many numbers in the set are relatively prime to m ( $gcd=1$ ), finding  $\Phi(m)$  is computationally easy if factorization of m is known

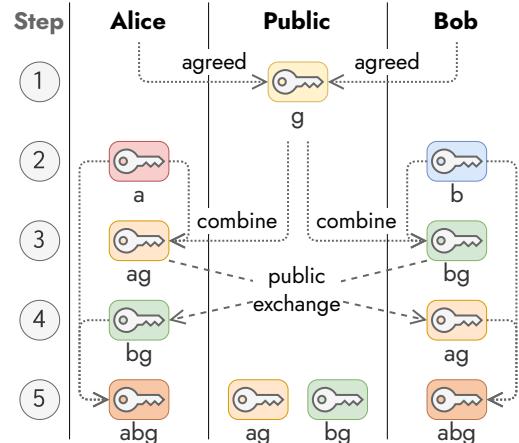
## Key Distribution Problem

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

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

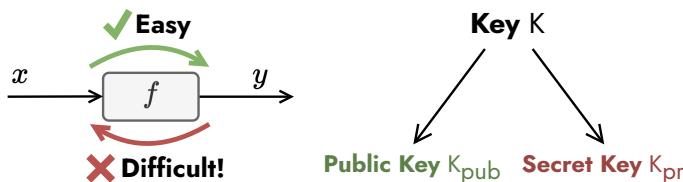


## Diffie Hellman YouTube 1, YouTube 2

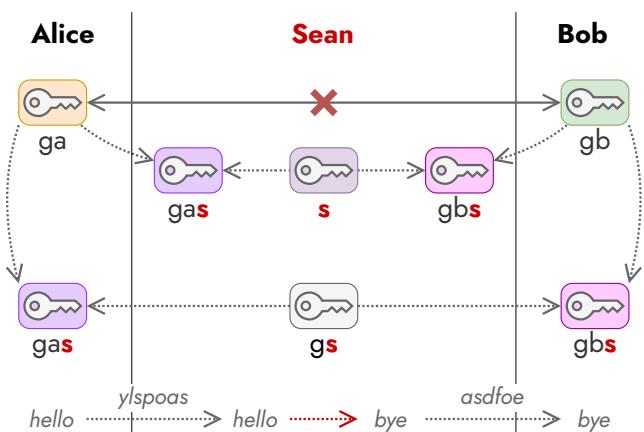


## Public Key Cryptography YouTube

- Key besteht aus **public** und **private** Teile.
  - **private**: Schlüssel ist nur mir bekannt (**don't share!**)
  - **public**: Schlüssel ist öffentlich verfügbar
- Was mit dem **private** key verschlüsselt wird kann mit dem **public** key entschlüsselt werden und vice versa
- Löst das Key Distribution System! **Aber** ist sehr Recheneintensiv!



## 🔥 Man-In-The-Middle Attack YouTube



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.

### Vorbereitung:

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

### Ausführung:

- Alice**
- 1 Wählen eines zufälligen **private** Key **a**
  - 2 **Public** Key berechnen:  $k_A = g^a \text{ mod } p$
  - 3 Key **k<sub>A</sub>** veröffentlichen

- Bob**
- 1 Wählen eines zufälligen **private** Key **b**
  - 2 **Public** Key berechnen:  $k_B = g^b \text{ mod } p$
  - 3 Key **k<sub>B</sub>** veröffentlichen

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

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

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

$$\begin{aligned} a^x \text{ mod } p &= k_{AP} \text{ mod } p \\ a^x &\equiv k_{AP} \quad \text{Un} \end{aligned}$$

## Beispiel

- Set-up by sender Alice or receiver Bob:
  - Choose a small prime  $p = 29$
  - Choose an integer  $a \in \{2, 3, \dots, p-2\} = 2$
  - Publish  $p = 29$  and  $a = 2$
- Alice: Choose random private key  $a=x$ ,  
Compute public key  $k_{AP} = 2^x \bmod 29 = 3$   
Publish public key  $k_{AP}=3$
- Bob: Choose random private key  $b=y$ ,  
Compute public key  $k_{BP} = 2^y \bmod 29 = 7$   
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)  
 $x = \log_a k_{AP} \bmod p$  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 !!!

## Beispiel

- Setup Bob:
  - Choose (far too small) primes  $p = 3$  and  $q = 11$
  - Compute  $n = p \cdot q = 33$
  - $\Phi(n) = (p-1) * (q-1) = (3-1) * (11-1) = 20$
  - Choose  $e = 3$
  - Publish  $33, 3$  ↘ keine Komma Zahl
  - Private key  $d \equiv e^{-1} \equiv 7 \bmod 20$  ( $e \cdot d \bmod 20 = 1$ )
- 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  $(s^{ed} \cdot y)^d = s^{ed} \cdot x^{ed} = s \cdot x \bmod n$   
=> 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.

## RSA is malleable

**The Problem:** Textbook RSA allows attackers to manipulate encrypted values without knowing the private key.

### How it Works:

- Attacker has original ciphertext:  $c = m^e \bmod n$
- Attacker chooses multiplier:  $s$
- Attacker computes:  $c' = c \cdot s^e \bmod n$
- When decrypted:  $(c')^d = m \cdot s \bmod n$

### The Formula:

$$c' = c \cdot s^e \bmod n (c')^d = (c \cdot s^e)^d = c^d \cdot s^{ed} = m \cdot s \bmod n$$

**Result:** The attacker successfully transforms an encryption of  $m$  into an encryption of  $m \cdot s$  without knowing  $m$  or the private key.

**Real Impact:** An attacker can multiply any encrypted value (money amounts, user IDs, etc.) by any chosen factor.

**Defense:** Use padding schemes like PKCS#1 OAEP that add structured randomness, making it virtually impossible for modified ciphertexts to decrypt to valid padded messages.

## ! 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.

## Elgamal Encryption Scheme

Erweiterung von DHKE → Ähnlicher Ablauf wie bei DHKE, einfach wird ein neuer 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):
  - Choose a large prime  $p$
  - Choose an integer  $a \in \{2, 3, \dots, p-2\}$
  - 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  $y = x \cdot k_{AB} \bmod p$   
Send  $y$  and  $k_{AP}$  to Bob
- Bob: Compute session key  $k_{AB} = (k_{AP})^b \bmod p$   
Decrypt  $y$  with  $x = y \cdot k_{AB}^{-1} \bmod p$

- + Alice can precompute many keys  $k_{AP}$  (and send them to Bob from him to precompute  $k_{AB}$ )
- Bandwidth requirement\*2 ( $y$  and  $k_{AP}$  for each block)

## RSA

### Vorbereitung:

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

### Ausführung:

**Alice** Verschlüsselt  $y = x^e \bmod n$

**Bob** Verschlüsselt  $x = y^d \bmod n$

**Oskar** Muss Faktor  $n$  aufwenden, um  $d$  zu erhalten

## Elliptic-Curve-Cryptography (ECC)

Die Elliptic-Curve-Cryptography ist einer vieler Ansätze, um einen Public Key zu generieren. Dieser Ansatz verwendet die Struktur einer elliptischen 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.

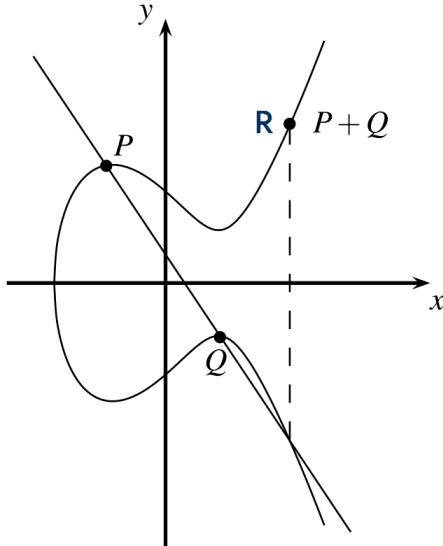
## Number Lists

Die Modulo funktion  $\bmod$  in vorhergehenden Algorithmen ist auch eine Kurve (Kreis von 0 bis  $n$ ), jedoch eine immer wieder wiederholende liste an Integern. Dieses wiederholen

verschleiert die ausgangsvariable (z.B.  $g^b$ ) und macht so die berechnung von  $b$  extrem schwierig. Der Ansatz über ECC ist noch komplizierter (und sicherer -> weniger bits für selbe Sicherheit).

$$\begin{aligned} R(x_3, y_3) &= P(x_1, y_1) + Q(x_2, y_2) \\ s &= \text{tangent} = (3 \cdot x_1^2 + a)/(2 \cdot y_1) \bmod p \\ x_3 &= s^2 - x_1 - x_2 \bmod p \quad \text{wie bei } P \neq Q \\ y_3 &= s \cdot (x_1 - x_3) - y_1 \bmod p \quad \text{wie bei } P \neq Q \end{aligned}$$

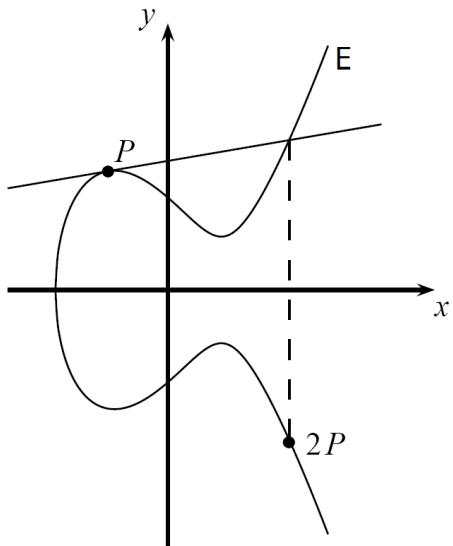
### Addition $P \neq Q$



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

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

### Addition $P = Q$



### ! Trapdoor: Double-and-Add for $d \cdot P$

```
• Double-and-Add Algorithm for  $d \cdot P$ :  

T=P  

FOR i = t-1 DOWNTO 0 // t #bits of  $d_2$   

    T = T+T mod n  

    IF  $d_i = 1$  // biti of  $d_2$   

        T = T + P mod n // to prevent power side channel attack add: ELSE X=T+P mod n  

RETURN (T)
```

### Elliptic-Curve Diffie-Hellmann Key Exchange (ECDH)

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

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\*\* ( $n, e$ )  
 $x' = \text{verif}_{K_{\text{Pub}}}(s) = s^e \bmod n$

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  $2^{159} < q < 2^{160}$   $\quad // (p-1)/q \in N$
3. Find an integer  $a$  with  $\text{ord}(a)=q$   
 $\text{ord}(a)$  is the smallest positive integer  $k$  with  $a^k \equiv 1 \pmod q$   $// S.93,211$
4. Choose a random integer as private key  $d$  with  $0 < d < q$
5. Compute  $\beta \equiv a^d \pmod p$
6. Publish  $K_{\text{pub}} = (p, q, a, \beta)$

### DSA signature generation:

Given: message  $x$ , private key  $d$  and  
public key  $(p, q, a, \beta)$

1. Choose an integer as random ephemeral (flüchtig) key  $k_E$  with  $0 < k_E < q$
2. Compute  $r \equiv (a^{k_E} \pmod p) \pmod q$
3. Compute  $s \equiv (\text{SHA}(x) + d \cdot r) k_E^{-1} \pmod q$  =>  
signature  $(r, s)$

### DSA signature verification

Given: message  $x$ , signature  $s$  and public key  $(p, q, a, \beta)$

1. Compute auxiliary value  $w \equiv s^{-1} \pmod q$
2. Compute auxiliary value  $u_1 \equiv w \cdot \text{SHA}(x) \pmod q$
3. Compute auxiliary value  $u_2 \equiv w \cdot r \pmod q$
4. Compute  $v \equiv (a^{u_1} \cdot \beta^{u_2} \pmod p) \pmod q$
5. Valid signature:  $v = r \pmod q$ ;  
invalid signature:  $v \neq r \pmod q$

#### Sign Bob:

- Compute hash of message  $H(x) = 26$
1. Choose ephemeral key  $k_E = 10$
  2.  $r = (3^{10} \pmod 59) \pmod 29 = 20$
  3.  $s = (26 + 7 \cdot 20 \cdot 3) \equiv 5 \pmod 29$   $// k_E \cdot k_E^{-1} = 1 = 10 \cdot 3 = 30 \equiv 1 \pmod{29}$
  4. Send  $(x, r, s) = (x, 20, 5)$

#### Verify Alice:

1.  $w \equiv 5^{-1} \equiv 6 \pmod{29}$   $// (5 \cdot 6) \pmod{29} = 30 \pmod{29} = 1$
2.  $u_1 \equiv 6 \cdot 26 \equiv 11 \pmod{29}$
3.  $u_2 \equiv 6 \cdot 20 \equiv 4 \pmod{29}$
4.  $v = (3^{11} \cdot 6^4 \pmod{59}) \pmod{29} = 20$   
 $v \equiv r \pmod{29} \rightarrow$  valid signature

### Note

- 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)

**Problem:** Jeder benötigt den key von jedem.

Remember the  $n^2$  Key Distribution Problem:

for  $n$  participants  $n(n-1) \approx n^2$  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  $\text{sigKCA}(k_{\text{pub}}, \dots)$

**Trust:** Must be trusted by all users.

**Certificate:** Contains public key and user ID with CA's digital signature (e.g.,  $\text{Cert}_{Alice} = (k_{\text{pub}}, ID_{Alice}, \text{sig}_{KCA}(k_{\text{pub}}, ID_{Alice}))$ ), 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).

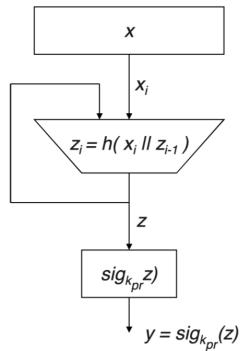
**Weakness:** Initial transmission of CA public key is not authenticated.

**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

### Hash Functions

- Instead of signing the whole message (e.g. DSH), sign only a digest (=hash) => much faster
- Hash functions  $h(x)$  have **no key** and are **public**
- Preimage resistance:**  
For a given output  $z$ , it is impossible to find an input  $w$  with  $h(w)=z$ ,  $h(x)$  is **one way**
- Second preimage resistance (collision):**  
Given  $x_1$ , and thus  $h(x_1)$ , it is infeasible to find any  $x_2$  such that  $h(x_1)=h(x_2)$ .
- Collision resistance:** It is infeasible to find  $x_1 \neq x_2$  such that  $h(x_1)=h(x_2)$
- How hard is it to find a collision with a probability of 0.5 (brute force)?  
**(Birthday paradox:** two humans with same birthday in the room?  
Only 23 persons) => Hash functions need an output size of **≥160 bits** to withstand ~ $2^{80}$  tries **Make it hard through high computational need**
- Hash functions are based on block ciphers or dedicated algorithms (MD5 family, SHA-1, RIPE-MD160)

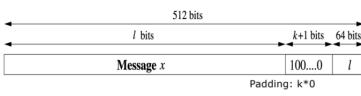


## SHA-1

Vorwärts rechnen: easy  
Rückwärts rechnen: Fast unmöglich

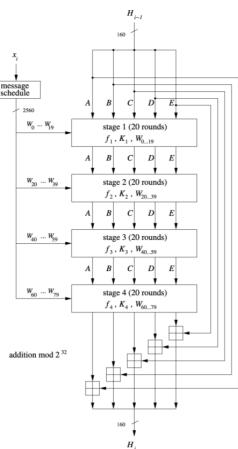
### Secure Hash Algorithm SHA-1 (of MD-4 family)

- 4 Stages with 20 rounds, 32bit operations, Output: **160-bit Hash**
- Merkle-Damgård construction
- Widley used (despite known weaknesses)
- Message schedule:**  
for ( $j=0$ ;  $j<80$ ;  $j+=$ )  
  if  $j < 16$ :  $W_j = x_j$ ,  
  if  $j > 15$ :  $W_j = \text{rotl}(W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-3}, 1)$   
// 5: 512->2560b  
// 80\*32b= 2560b  
// 16\*32b= 512b  
// 64\*32b=4x512b
- Initial H:  $H_0 = H_0^0 || H_0^1 || H_0^2 || H_0^3 || H_0^4$   
=(0x67452301, 0xEFCDAB89, 0x98BADCFF, 0x10325476, 0xC3D2E1F0)
- Padding of last 512 bit block:



HSLU 28. März 2025 ©M.Thalheim

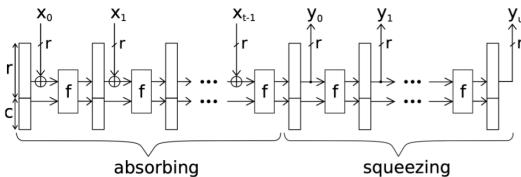
Understanding Cryptography by Christof Paar and Jan Pelzl



## SHA-3 (Keccak)

Bit's werden mit jeder Iteration verstrichen (ein Bit besteht aus hunderten Bits) -> Rückwärts nicht rechenbar.

b =	r    c	Security level	Output
1600	1152	448	112
1600	1088	512	128
1600	832	768	192
1600	576	1024	256
			512



## weitere Begriffe

**Keyspace** Anzahl möglichen & relevanten Keys

**Brute Force** Alle Keykombination versuchen (Erfolg  $\approx$  Keyspace/2)

**Frequency Analysis** Gewisse Zeichen(kombinationen) werden häufiger verwendet

## Reliability

## Vulnerability

## Boot

## Memory

## Execution

## Fault Handling