Verlässliche Echtzeitsysteme - Übungen

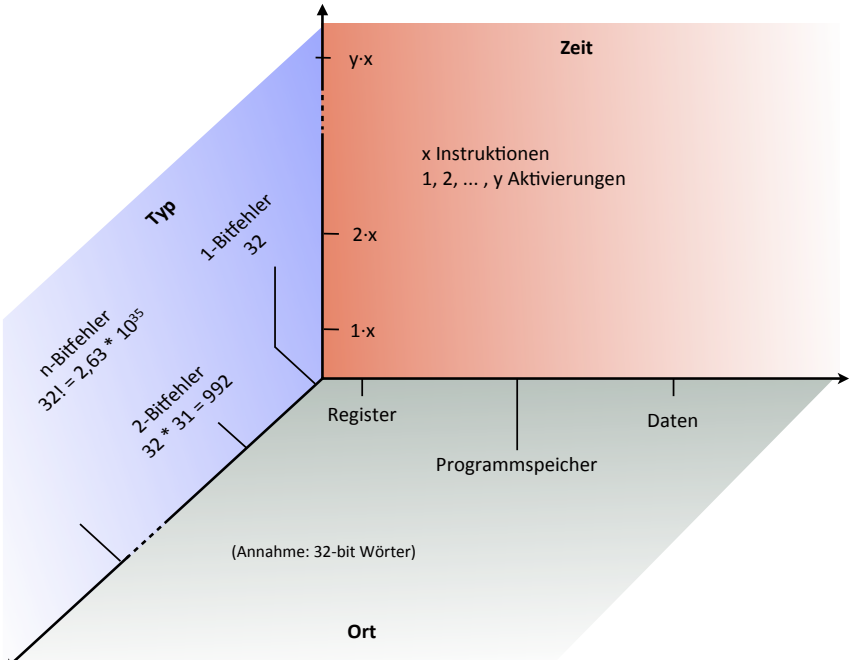
Arbeiten mit Binärdateien und Assembler, Codierung

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Systemsoftware)
https://sys.cs.fau.de

Fehlerraum


Zeit
y·x
x Instruktionen
1, 2, ... , y Aktivierungen
2·x
1·x
Typ
1-Bitfehler
32
n-Bitfehler
32! = 2,63 * 10^35
2-Bitfehler
32 * 31 = 992
Register
Daten
Programmspeicher
(Annahme: 32-bit Wörter)
Ort

C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
  static int s = 3;
  int x, y;
  char* p =
       malloc(100);
}
```

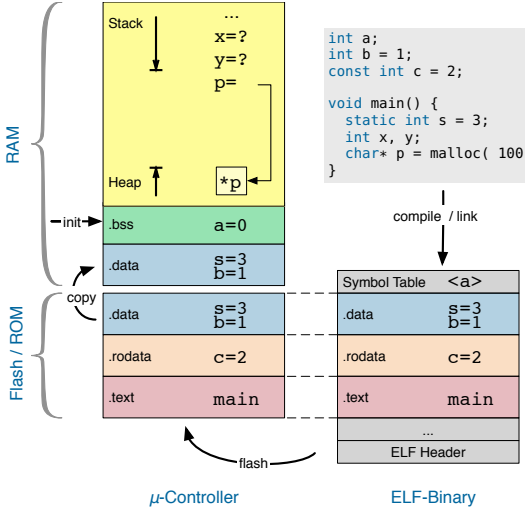
Assembler-Code

```
4004f0 <main>:
  4004f0: push   %rbp
  4004f1: mov    %rsp,%rbp
  4004f4: sub    $0x10,%rsp
  4004f8: movabs $0x64,%rdi
  400502: callq  4003e0 <malloc@plt>
  400507: mov    %rax,-0x10(%rbp)
  40050b: add    $0x10,%rsp
  40050f: pop    %rbp
  400510: retq
```

Wo können Datenfehler auftreten?

1. RAM: `-0x10(%rbp)`
   - ⇝ Stack, globale Daten, Heap, (Programmcode)
2. Allgemeine CPU-Register: `%rsp`
3. Sonstige CPU-Register: `%rip`, `%rflags`

Speicherorganisation und Sektionsgrößen




```c
int a;                          // a: global, uninitialized
int b = 1;                      // b: global, initialized
const int c = 2;                // c: global, const

void main() {
  static int s = 3;             // s: local, static, initialized
  int x, y;                     // x: local, auto; y: local, auto
  char* p = malloc( 100 );      // p: local, auto; *p: heap (100 byte)
}
```

Quellprogramm

```
$ size --format=berkeley a.out
text data bss  dec hex filename
2592    8   4 2604 a2c     a.out
```

Sektionsgrößen in Byte,
„`text`“ umfasst hier: `.text` + `.rodata`

Umgang mit Assembly-Code I

nm: Ausgabe der Symboltabelle

```
0000000000004028 D var_initialized
...
0000000000201028 B __bss_start
0000000000201028 b var_uninitialized
...
000000000000061a T main
0000000000000580 t register_tm_clones
0000000000000510 T _start
...
000000000000066d t int max<int,0u>(int,int)
000000000000067c t int max<int,1u>(int,int)
```

- Nützliche Optionen
  - `-C, --demangle`: Dekodieren der C++-Namensmangelung:

    `_Z3maxIiLj0EET_S0_S0_` ⇒ `int max<int, 0u>(int, int)`
  - `-S, --print-size`: Ausgabe der Symbolgrößen

    `000000000000061 a 00000028 T main`

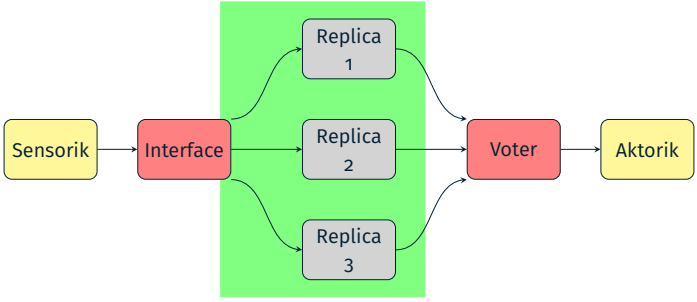
Umgang mit Assembly-Code II

`objdump`: Ausgabe von Informationen über Objektdateien

```
int main(){
 4007cd:   55                    push   %rbp
 4007ce:   48 89 e5              mov    %rsp,%rbp
 4007d1:   48 83 ec 10           sub    $0x10,%rsp
int a = max<int>(23U, 42);
 4007d5:   be 2a 00 00 00        mov    $0x2a,%esi
 4007da:   bf 17 00 00 00        mov    $0x17,%edi
 4007df:   e8 04 01 00 00        callq
                                 4008e8 <_Z3maxIiET_S0_S0_>
 4007e4:   89 45 fc              mov    %eax,-0x4(%rbp)
std::cout << a << "\n";
 4007e7:   8b 45 fc              mov    -0x4(%rbp),%eax
 ...
```

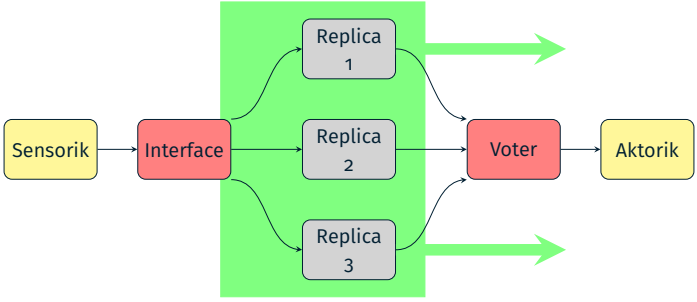
- Nützliche Optionen
  - `-S`: Ausgabe von Quell-Code im Assembly-Code (Debug-Symbole notwendig)
  - `-D`: alle Sektionen disassemblieren

Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

Klassische “Triple Modular Redundancy” (TMR)



Redundanzbereich

Ausschließlich Replikatausführung

- ⇝ Erweiterung der Ausgangsseite mit Informationsredundanz
- ⇝ Mehrheitsentscheid über codierte Prüfsumme

Erweiterte arithmetische Codierung

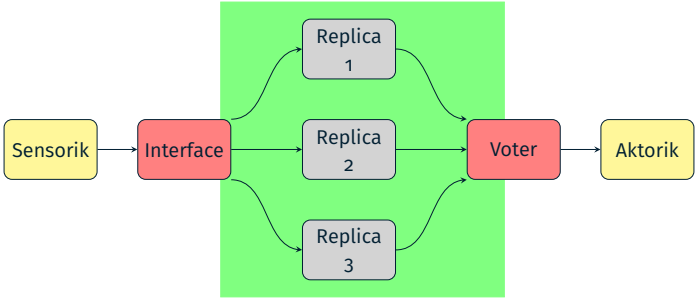
nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems”

- Arithmetisch codierter Wert V_C
- Ausgangswert
- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel

Wertebereichseinschränkungen

- Schlüssel A sollte so groß wie möglich sein:
  - $\rightsquigarrow$ Möglichst geringe Restfehlerwahrscheinlichkeit ($P = 1/A$)
- Wertebereich des dynamischen Zeitstempels
  - $D = \{x | x \in \mathbb{N}_0 \wedge x \leq D_{max}\}$
  - Zeitstempel darf überlaufen: $D_{max} + 1 = 0$
- Für jede Signatur $B_* \in \mathbb{N}$ muss dann gelten
  - $B_* + D_{max} < A$
  - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss größer D_{max} sein: $\forall i, j : |B_i - B_j| > D_{max}$
  - Sämtliche Distanzen zwischen jeweils zwei Signaturen müssen unterschiedlich sein: $\forall i : \forall j, k : |B_i - B_j| = |B_k - B_j| \Rightarrow i = k$

Erweiterung I – codierte Ausgangswerte



- Replikate liefern arithmetisch codierte Ergebnisse
- Mehrheitsentscheid auf codierten Prüfsummen
- Übertragung codierter Ergebnisse

EAN Vergleichsoperator

Vereinfachung für diese Übungsaufgabe

- Kein Zeitstempel

- Voting basiert auf codierter Vergleichsoperation:

$\rightsquigarrow X_C = Y_C \Rightarrow X * A + B_X = Y * A + B_Y$

- Im fehlerfreien Fall gilt:

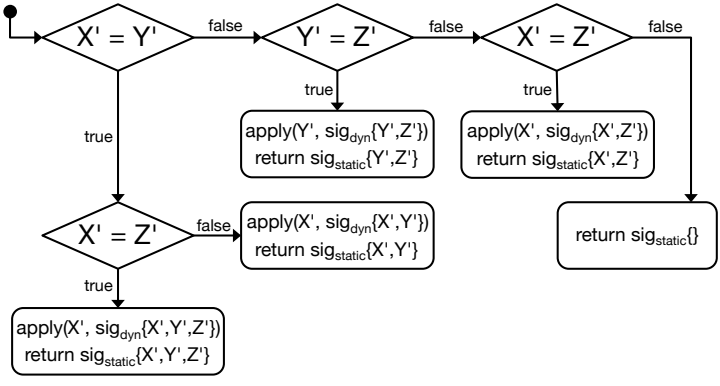
$X = Y$, $A = A$ aber $B_X \neq B_Y$!

  - *Rohwerte* sind identisch
  - Schlüssel ist per Definition identisch
  - Signaturen sind unterschiedlich (aber konstant!)

Bestimmung der Gleichheit durch Differenzbildung:

$\rightsquigarrow X_C - Y_C = B_X - B_Y = const.$

Codierter Mehrheitsentscheid



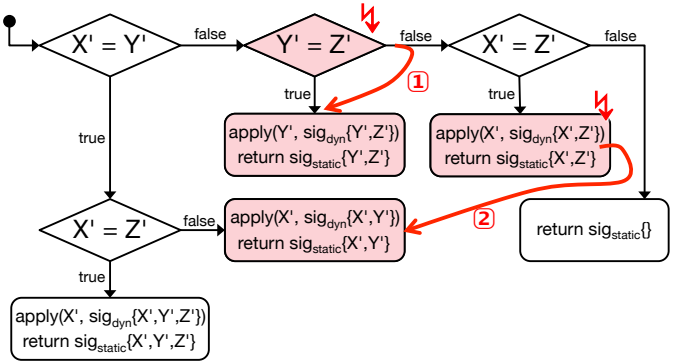
- Bestimmung von dynamischer und statischer Signatur:
  - → $sig_{dyn}(X', Y') : X' = Y' \Rightarrow X' - Y'$
  - → $sig_{static}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$
- Für die Signaturen muss gelten: $B_X > B_Y > B_Z$

Codierter Mehrheitsentscheid (Forts.)

1. Vergleichsoperation wird durchgeführt (z. B. $X' = Y' \wedge X' = Z'$)
   - Berechnung von sig_{dyn}
   - Vergleich mit sig_{static}
2. Verzweigungsentscheidung wird nachberechnet:
   - Wiederholte (redundante) Berechnung von sig_{dyn}
   - Erneuter Vergleich: $sig_{dyn} = sig_{static}$
   - Addiere sig_{dyn} (*apply*) zum gewählten Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
   - Aktor wählt entsprechendes Replikatergebnis
   - Führt inverse Operation zu *apply* durch

Im Voter wurde die *dynamisch berechnete Signatur der Verzweigungsentscheidung* hinzu addiert. Im Aktor wird mit der entsprechenden *konstanten Signatur zurückgerechnet*.

Codierter Mehrheitsentscheid - Fehlerfall



1. Falsche Verzweigungsentscheidung: ($Y' \neq Z'$)
   - Y' wird als korrekt angenommen, sig_{dyn} wird berechnet
   - allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
   - Fehler wird vor dem *apply* erkannt
2. Falscher (plötzlicher) Sprung

Ausgangslage

- Wähle $A = 601$
- Initiale Ergebniskodierung in den Replikaten:

	X	Y	Z
Wert	7	5	7
B	37	23	5
Kodiert	$X' = A * 7 + 37$ $= 4244$	$Y' = A * 5 + 23$ $= 3028$	$Z' = A * 7 + 5$ $= 4212$

- Berechnung der statischen Signaturen **vorab, statisch**:

$$sig_{static}\{X', Y', Z'\} = (B_X - B_Y) + (B_X - B_Z) = 46$$
$$sig_{static}\{X', Y'\} = (B_X - B_Y) = 14$$
$$sig_{static}\{Y', Z'\} = (B_Y - B_Z) = 18$$
$$sig_{static}\{X', Z'\} = (B_X - B_Z) = 32$$

Regulärer Durchlauf

- Das eigentliche Voting geschieht dann zur Laufzeit:

1. $X' = Y'$?

$$X' - Y' \stackrel{?}{=} sig_{static}\{X', Y'\} \Leftrightarrow 4244 - 3028 = 1216 \stackrel{?}{=} 14 \Leftrightarrow \textit{false}$$

2. $Y' = Z'$?

$$Y' - Z' \stackrel{?}{=} sig_{static}\{Y', Z'\} \Leftrightarrow 3028 - 4212 = -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$$

3. $X' = Z'$?

$$X' - Z' \stackrel{?}{=} sig_{static}\{X', Z'\} \Leftrightarrow 4244 - 4212 = 32 \stackrel{?}{=} 32 \Leftrightarrow \textit{true}$$

4. Berechnung der dynamischen Signaturen zur Laufzeit:

$$sig_{dyn}\{X', Z'\} = (X' - Z') = (4244 - 4212) = 32$$

Regulärer Durchlauf (II)

5. $sig_{dyn}\{X', Z'\} \stackrel{?}{=} sig_{static}\{X, Z\} \iff 32 \stackrel{?}{=} 32 \iff \textit{true}$
6. $X' = apply\left(X', sig_{dyn}\{X', Z'\}\right) = 4276$
7. $B_E \leftarrow sig_{static}\{X', Z'\} = 32$
8. $return(32)$
9. Nachschlagen der zugrundeliegenden Variable mittels $B_{dyn} = 32$ (erste Variable der Konsensmenge), basierend auf den vorberechneten statischen Werten:

$$\left(result_{variable}, result_{extrasignature}\right) \leftarrow (X', B_X)$$

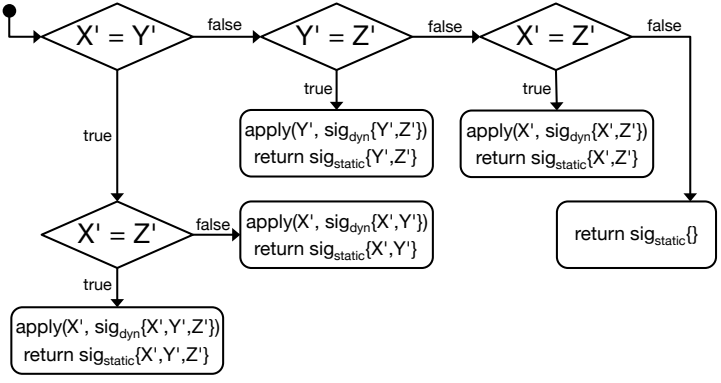
10. $inv\_apply\left(result_{variable}, B_{dyn}\right) = inv\_apply\left(4276, 32\right) = 4276 - 32 = 4244$

Regulärer Durchlauf (III)

11. Signaturverifikation:

$$check(4244, A, B_X)\text{: } \frac{4244 - B_X}{A} = 7 \text{ Rest: } 0$$

12. Ergebnis erfolgreich dekodiert: 7



Es hat somit eine erfolgreiche Einigung auf die Konsensmenge $\{X', Z'\}$ stattgefunden.

Ausgangslage (unverändert)

- Wähle $A = 601$
- Initiale Ergebniskodierung in den Replikaten:

	X	Y	Z
Wert	7	5	7
B	37	23	5
Kodiert	$X' = A * 7 + 37$ $= 4244$	$Y' = A * 5 + 23$ $= 3028$	$Z' = A * 7 + 5$ $= 4212$

- Berechnung der statischen Signaturen **vorab, statisch**:

$$sig_{static}\{X', Y', Z'\} = (B_X - B_Y) + (B_X - B_Z) = 46$$
$$sig_{static}\{X', Y'\} = (B_X - B_Y) = 14$$
$$sig_{static}\{Y', Z'\} = (B_Y - B_Z) = 18$$
$$sig_{static}\{X', Z'\} = (B_X - B_Z) = 32$$

Fehlerszenario ①

1. $X' = Y'$?

$$X' - Y' \stackrel{?}{=} sig_{static}\{X', Y'\} \Leftrightarrow 4244 - 3028 = 1216 \stackrel{?}{=} 14 \Leftrightarrow \textit{false}$$

2. $Y' = Z'$?

$$Y' - Z' \stackrel{?}{=} sig_{static}\{Y', Z'\} \Leftrightarrow 3028 - 4212 = -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$$

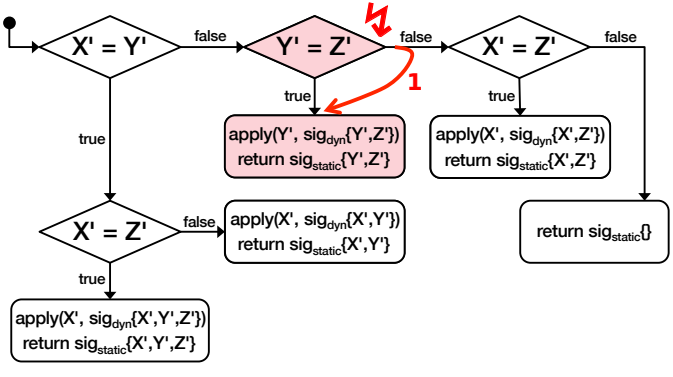
3. Hier tritt nun der Operatorfehler ein, die falsche Verzweigung ① wird aus gewählt
4. Berechnung der dynamischen Signaturen zur Laufzeit:

$$sig_{dyn}\{Y', Z'\} = (Y' - Z') = (3028 - 4212) = -1184$$

5. $sig_{dyn}\{Y', Z'\} \stackrel{?}{=} sig_{static}\{Y, Z\} \Leftrightarrow -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$

Fehlerszenario ① (II)

6. Fehler erfolgreich detektiert: Vergleich zwischen sig_{dyn} und sig_{static} für Konsensmenge $\{Y', Z'\}$ fehlgeschlagen



Ausgangslage (unverändert)

- Wähle $A = 601$
- Initiale Ergebniskodierung in den Replikaten:

	X	Y	Z
Wert	7	5	7
B	37	23	5
Kodiert	$X' = A * 7 + 37$ $= 4244$	$Y' = A * 5 + 23$ $= 3028$	$Z' = A * 7 + 5$ $= 4212$

- Berechnung der statischen Signaturen **vorab, statisch**:

$$sig_{static}\{X', Y', Z'\} = (B_X - B_Y) + (B_X - B_Z) = 46$$
$$sig_{static}\{X', Y'\} = (B_X - B_Y) = 14$$
$$sig_{static}\{Y', Z'\} = (B_Y - B_Z) = 18$$
$$sig_{static}\{X', Z'\} = (B_X - B_Z) = 32$$

Fehlerszenario ②

1. $X' = Y'$?

$$X' - Y' \stackrel{?}{=} sig_{static}\left\{X', Y'\right\} \Leftrightarrow 4244 - 3028 = 1216 \stackrel{?}{=} 14 \Leftrightarrow \textit{false}$$

2. $Y' = Z'$?

$$Y' - Z' \stackrel{?}{=} sig_{static}\left\{Y', Z'\right\} \Leftrightarrow 3028 - 4212 = -1184 \stackrel{?}{=} 18 \Leftrightarrow \textit{false}$$

3. $X' = Z'$?

$$X' - Z' \stackrel{?}{=} sig_{static}\left\{X', Z'\right\} \Leftrightarrow 4244 - 4212 = 32 \stackrel{?}{=} 32 \Leftrightarrow \textit{true}$$

4. Berechnung der dynamischen Signaturen zur Laufzeit:

$$sig_{dyn}\left\{X', Z'\right\} = (X' - Z') = (4244 - 4212) = 32$$

Fehlerszenario ②

5. $sig_{dyn}\{X', Z'\} \stackrel{?}{=} sig_{static}\{X, Z\} \Leftrightarrow 32 \stackrel{?}{=} 32 \Leftrightarrow \textit{true}$
6. $X' = apply\left(X', sig_{dyn}\{X', Z'\}\right) = 4276$
7. Hier tritt nun der Kontrollflussfehler ② ein
8. $B_E \leftarrow sig_{static}\{X', Y'\} = 14$
9. $return(14)$
10. Nachschlagen der zugrundeliegenden Variable mittels $B_{dyn} = 14$ (erste Variable der Konsensmenge), basierend auf den vorberechneten statischen Werten:

$$\left(result_{variable}, result_{extrasignature}\right) \leftarrow (X', B_X)$$

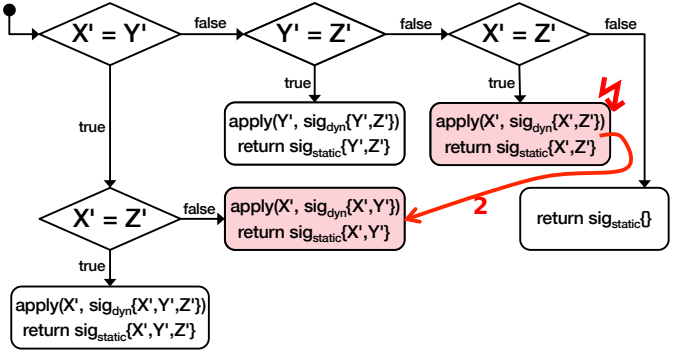
11. $inv\_apply\left(result_{variable}, B_{dyn}\right) = inv\_apply\left(4276, 14\right) = 4276 - 14 = 4262$

Fehlerszenario ② (II)

12. Signaturverifikation:

$$check(4262, A, B_X): \frac{4262 - B_X}{A} = 7 \text{ Rest: } 18$$

13. Fehler erfolgreich detektiert: Dekodieren schlägt fehl



Hinweise zur Aufgabenbearbeitung

Analyse der Toleranzmaßnahmen

- Fehlerraumanalyse
  - Einfluss des Speichers
  - Einfluss der Laufzeit

- Aufrufgraph

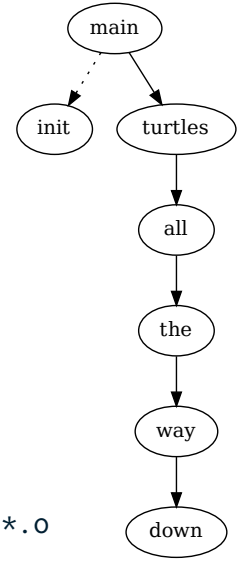
  `make callgraph_{baseline,tmr,ean}`

- ELF

  `build/{baseline,tmr,ean}.elf`

- Objektdateien

  `build/CMakeFiles/{baseline,tmr,ean}.dir/src/*.o`



Aufgabenstellung

Aufgabe

- Absichern des Voters per EAN mittels CoRed-Ansatz
- Berechnungen mit codierten Werten
  - Berücksichtigung der (statischen) Signaturen

  ⇝ Eigene Operationen mit konstanten Signaturwerten notwendig

  ⇝ bspw. eigenes `equals` anstatt ==