

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Codierung, Fehlerinjektion

Tobias Klaus, Florian Schmaus, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

19. April 2016



- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL*
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL*
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



- C99
- x86 bzw. x86-64, d. h.
 - vorzeichenbehaftete Integer als Zweierkomplement implementiert
 - char hat 8 Bit
 - short hat 16 Bit
 - int hat 32 Bit
 - long hat 32 Bit auf x86 und 64 Bit auf x86-64



Frage 10

Angenommen x hat Typ `int` und ist positiv. Ist $x \ll 1 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?



Frage 10

Angenommen x hat Typ `int` und ist positiv. Ist $x \ll 1 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Es darf nicht in das Vorzeichenbit hineinverschoben werden
- ⇒ nicht definiert für große Werte von x



Frage 11

Angenommen x hat Typ `int`. Ist $x \ll 31 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?



Frage 11

Angenommen x hat Typ `int`. Ist $x \ll 31 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Es darf nicht in das Vorzeichenbit hineinverschoben werden

⇒ funktioniert hier nur mit $x == 0$



Frage 12

Angenommen x hat Typ `int`. Ist $x << 32 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?



Frage 12

Angenommen x hat Typ `int`. Ist $x \ll 32 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Verschiebung um Bitbreite eines Datentyps nicht zulässig



- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL***
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL***
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



 https://www4.cs.fau.de/Lehre/SS16/V_VEZS/Uebung/Folien/Fail.pdf



```
#ifndef _include_fail_h
#define _include_fail_h

//Mark start of injection
volatile void __attribute__((noinline)) fail_start_trace(void);
//Mark end of injection
volatile void __attribute__((noinline)) fail_stop_trace(void);

//everything ok: valid code and right values
volatile void __attribute__((noinline)) fail_marker_positive(void);
//everything ok: valid code but wrong values
volatile void __attribute__((noinline)) fail_marker_negative(void);
//invalid code
volatile void __attribute__((noinline)) fail_marker_detected(void);

#endif /* _include_fail_h */
```

- Markierung Start/Ende für „Golden Run“
- Ziel: Minimierung Auftreten von fail_marker_negative



```
void cyg_user_start() {
    ...
    fail_trace_start() // Start Fehlerinjektion
    filter()
    vote()             // (codierter) Mehrheitsentscheid
    fail_trace_stop()  // Ende Fehlerinjektion

    actuate()          // Signaturen entfernen, Fehler überprüfen
    ...
}

void actuate() {
    ...
    if (checksum == expected_checksum){
        fail_marker_positive() // SDC behandelt :-
    }else{
        fail_marker_negative() // undetektierte SDC :-
    }
}
```



- Datei anlegen: `~/ .my.cnf`
- Folgende Daten eintragen:
`[client]`
`user=vezs20`
`password=XXXXXXXXXXXXXXXXXX`
`host=i4jenkins.cs.fau.de`
`database=vezs20`
- Zugangsdaten wurden verteilt



1. make trace

- Erstellung des „Golden Runs“
- Speicherung der Experimente in Datenbank
- Faltung des Fehlerraums

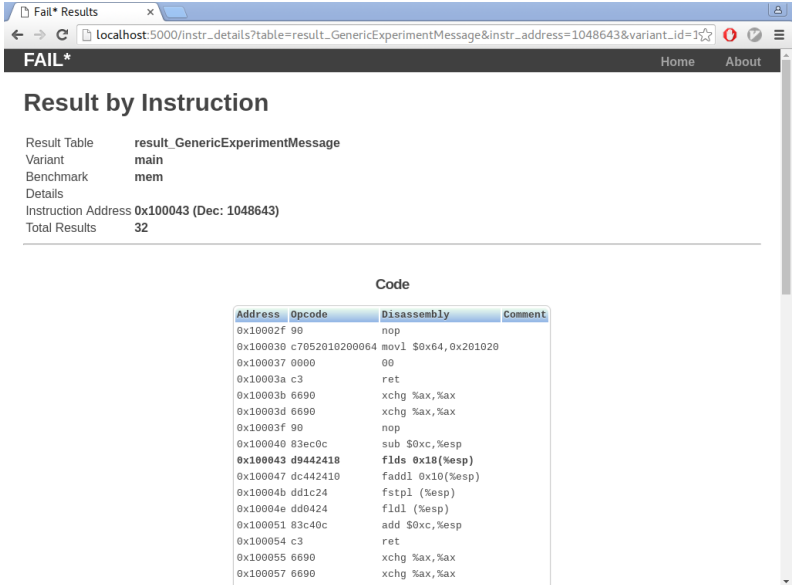
2. make inject

- Durchführung Fehlerinjektion
- Überblick der Ergebnisse in lokal angelegter Datei: `ean_result.csv`
 - 👉 Abschätzung des zeitlichen Aufwands

3. make resultbrowser

- Startet lokalen Webserver
- Ergebnisse per Webbrowser einsehbar





Fail* Results

localhost:5000/instr_details?table=result_GenericExperimentMessage&instr_address=1048643&variant_id=1

FAIL* Home About

Result by Instruction

Result Table **result_GenericExperimentMessage**
Variant **main**
Benchmark **mem**
Details
Instruction Address **0x100043 (Dec: 1048643)**
Total Results **32**

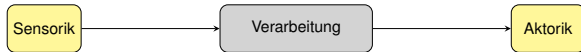
Code

Address	Opcode	Disassembly	Comment
0x10002f	90	nop	
0x100030	c7052010200064	movl \$0x64,0x201020	
0x100037	0000	00	
0x10003a	c3	ret	
0x10003b	6690	xchg %ax,%ax	
0x10003d	6690	xchg %ax,%ax	
0x10003f	90	nop	
0x100040	83ec0c	sub \$0xc,%esp	
0x100043	d9442418	flds 0x18(%esp)	
0x100047	dc442410	faddl 0x10(%esp)	
0x10004b	dd1c24	fstpl (%esp)	
0x10004e	dd0424	fldl (%esp)	
0x100051	83c40c	add \$0xc,%esp	
0x100054	c3	ret	
0x100055	6690	xchg %ax,%ax	
0x100057	6690	xchg %ax,%ax	

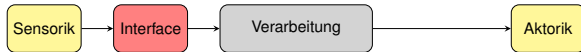
- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL*
- 3 Wiederholung Software-TMR**
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



Klassische “Triple Modular Redundancy” (TMR)



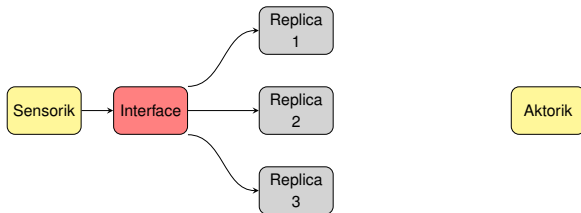
Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)



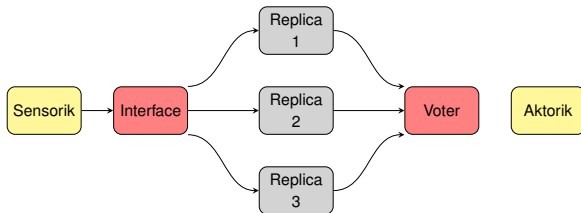
Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate

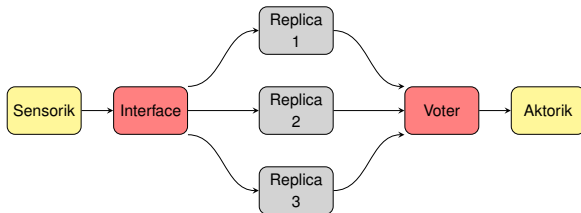


Klassische “Triple Modular Redundancy” (TMR)



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

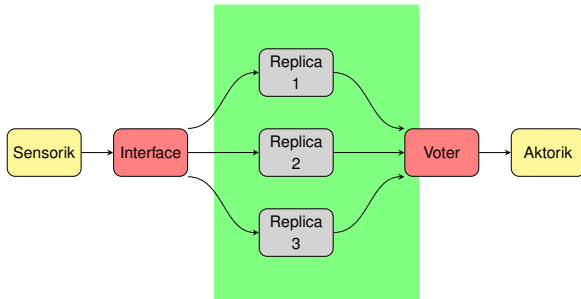
Klassische “Triple Modular Redundancy” (TMR)



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



Klassische “Triple Modular Redundancy” (TMR)

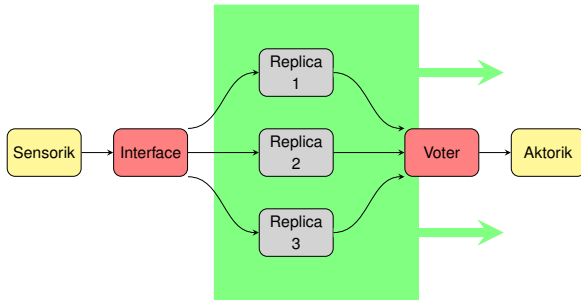


Redundanzbereich

Ausschließlich Replikatausführung.



Klassische “Triple Modular Redundancy” (TMR)



Redundanzbereich

Ausschließlich Replikatausführung.

- Erweiterung der Ausgangsseite mit Informationsredundanz
- Mehrheitsentscheid über codierte Prüfsumme



- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL*
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR**
- 5 Aufgabenstellung



nach Forin 1989: "Vital coded microprocessor principles and application for various transit systems" [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert


$$V_C = V$$



nach Forin 1989: "Vital coded microprocessor principles and application for various transit systems" [1]

- Arithmetisch codierter Wert V_C

- Ausgangswert

$$V_C = V * A$$

- Schlüssel

Bitfehlererkennung
(Restfehlerwahrscheinlichkeit
 $P = 1/A$)



nach Forin 1989: "Vital coded microprocessor principles and application for various transit systems" [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = V * A + B_V$$

- Schlüssel
- Variablenspezifische Signatur

Adressierungsfehlererkennung



nach Forin 1989: "Vital coded microprocessor principles and application for various transit systems" [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = V * A + B_V + D$$

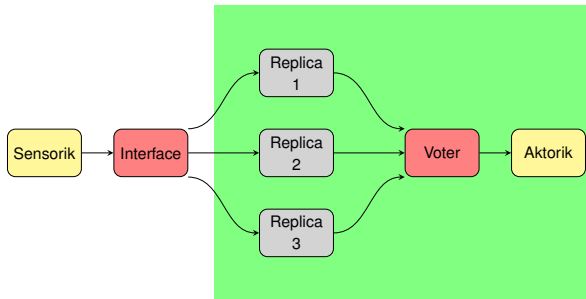
- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel

Erkennung
veralteter Daten



- Schlüssel A sollte so groß wie möglich sein:
 - ↪ 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_* muss dann gelten
 - $B_* + D_{max} < A$
 - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss kleiner D_{max} sein: $\forall i, j : |B_i - B_j| < D_{max}$





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

Für diese Übungsaufgabe:

- Kein Zeitstempel
- Nur Absicherung der Ausgangsseite!



- Voting basiert auf codierter Vergleichsoperation:

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

- Im fehlerfreien Fall gilt:

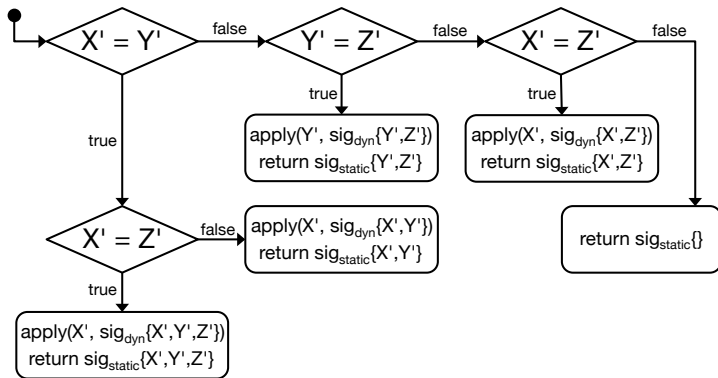
$X = Y$, $A = A$ aber $B_X \neq B_Y$!

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

Bestimmung der Gleichheit durch Differenzbildung:

→ $X_C - Y_C = B_X - B_Y = \text{const.}$





■ Bestimmung von dynamischer und statischer Signatur:

~ $\text{sig}_{\text{dyn}}(X', Y') : X' = Y' \Rightarrow X' - Y'$

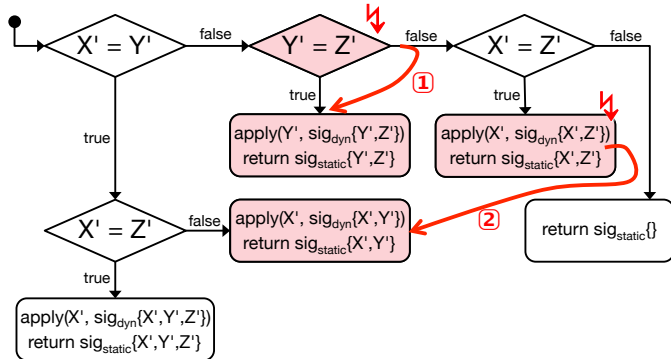
~ $\text{sig}_{\text{static}}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$



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}
 - Addiere sig_{dyn} (*apply*) zum gewählten Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
 - Akteur wählt entsprechendes Replikatergebnisse
 - Führt inverse Operation zu *apply* durch

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

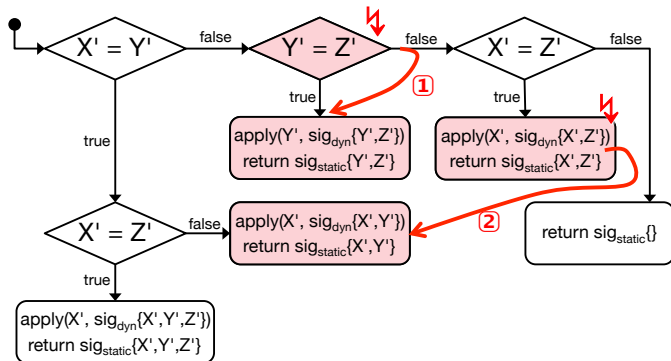




1. Falsche Verzweigungsentscheidung: ($Y' \neq Z'$)

- Y' wird als korrekt angenommen, sig_{dyn} wird berechnet
- allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
- Fehler wird bei der inversen Operation zu `apply` erkannt





2. Falscher (plötzlicher) Sprung

- X' wird als korrekt erkannt, sig_{dyn} wird erneut berechnet
- Ein fehlerhafter Sprung zu einem anderen Block führt zu einem inkonsistenten Rückgabewert $sig_{static}\{X', Z'\}$
- $sig_{dyn}\{X', Y'\} \neq sig_{static}\{X', Z'\}$ wird beim Decodieren erkannt



- 1 C-Quiz Teil IV
- 2 Fehlerinjektion mit FAIL*
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung**



Aufgabe

- Absichern des Voters per EAN mittels CoRed-Ansatz
 - Absichern des Akzeptanzmaskierers am Eingang
-
- Jedes Replikat hat genau einen Ausgabewert ($\text{integer} \mapsto \text{enc_t}$): eine codierte Prüfsumme des Ergebnisses
 - ↪ Festlegung für jeden der drei Ausgabewerte (X', Y', Z') jeweils *unterschiedliche* aber *konstante* Signatur (SIG_X, SIG_Y, SIG_Z)
 - Nutzung des nächstgrößeren Datentyps X für den ursprünglichen Wert X'
 - ↪ Wahl einer Zahl A mit möglichst großem Hamming-Abstand unter *Vermeidung* möglicher *Überläufe bei der Codierung*



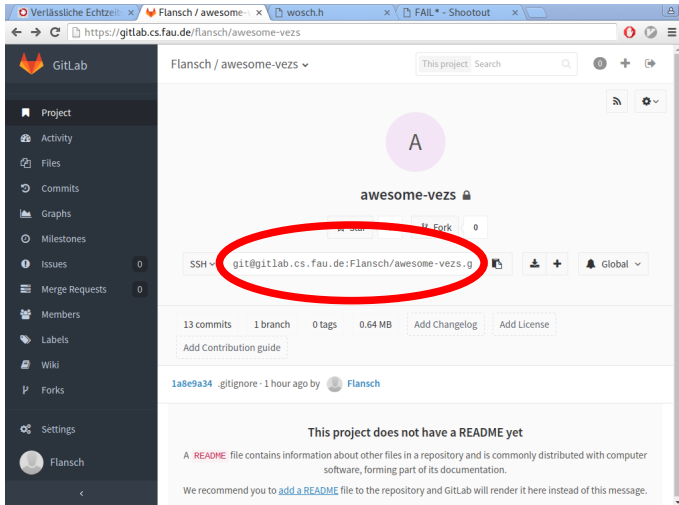
Für jede Operation zwischen zwei codierten Werten

ist eine eigene Operation mit konstanten Signaturwerten notwendig!

- Bestenliste wird automatisch auf Webseite veröffentlicht
- Wie werden Resultate generiert?

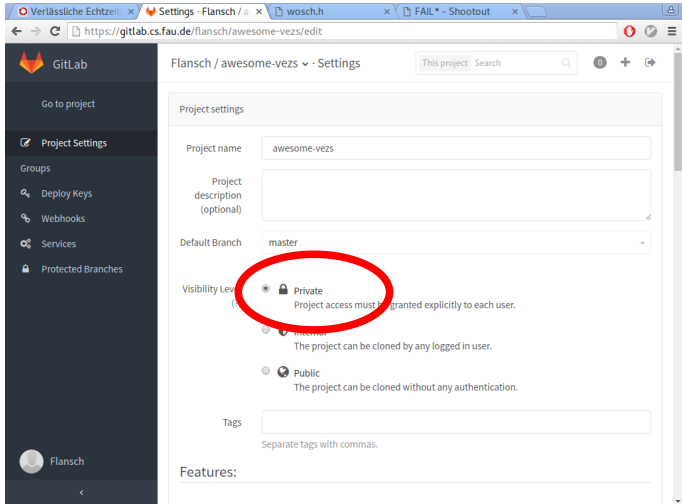


Repository Zugriff



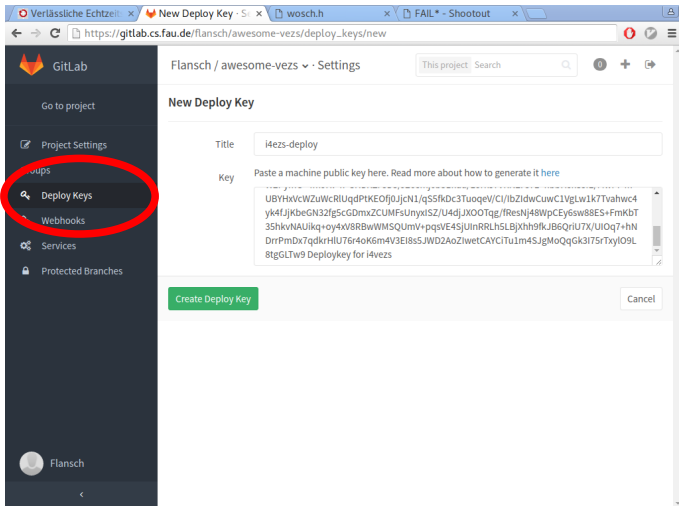
Repository URL mit Gruppe per Mail an Übungsleiter

Repository Einstellungen



Privates Repository konfigurieren

Repository Einstellungen – Deploy-Key hinterlegen



Repository Einstellungen – Überprüfung Deploy-Key

Verlässliche Echtzeit x Deployment Keys - Settings x wosch.h x FAIL* - Shootout x

https://gitlab.cs.fau.de/flansch/awesome-vezs/deploy_keys

GitLab

Go to project

Project Settings

Groups

Deploy Keys

Webhooks

Services

Protected Branches

Flansch

Flansch / awesome-vezs - Settings

This project Search

Deploy keys allow read-only access to the repository

+ New Deploy Key

Deploy keys can be used for CI, staging or production servers. You can create a deploy key or add an existing one

Enabled deploy keys for this project

i4ezs-deploy Remove

b9:40:31:68:27:d0:52:5b:97:1e:6a:f1:e5:2f:83

Flansch / awesome-vezs Created 8 minutes ago

Deploy keys from projects you have access to

Deploy keys from projects you have access to will be displayed here



42





Forin.

Vital coded microprocessor principles and application for various transit systems.

IFA-GCCT, pages 79–84, 1989.

