Messung der Schwierigkeit von Programmieraufgaben zur Kryptologie in Java

Konstantin Knorr[1]

Abstract: Systeme zur automatischen Bewertung von Programmieraufgaben (ABP) werden seit vielen Jahren erfolgreich in der Ausbildung von Informatikern eingesetzt, insbesondere in Zeiten verstärkter Online-Lehre. Kryptologie gilt bei vielen Studierenden aufgrund ihrer formellen und theoretischen Natur als schwer zugänglich. Das Verständnis kryptologischer Primitiven wie Ver- und Entschlüsselung oder Signatur und ihre Verifikation kann durch die Programmierung bzw. programmatische Anwendung gestärkt werden. Der Beitrag präsentiert eine Studie mit 20 Studierenden, 20 Aufgaben zur Kryptologie und ~300 JUnit-Testfällen, die über ein ABP-System ausgewertet wurden. Die Auswertung nach der Fehlerrate und dem Lösungszeitpunkt der kryptologischen Testfälle erlaubt die Identifikation von schweren Testfällen und zeigt u.a., dass Studierende weniger Fehler bei Substitutions- als bei Transpositionschiffren machen, symmetrische Chiffren leichter fallen als asymmetrische und dass Tests zu den Konstruktoren, Exceptions und Padding deutlich früher und besser gelöst wurden als Tests zu Signaturen und deren Verifikation.

Keywords: Bearbeitungszeit, Java, JUnit, Fehlerrate, Kryptologie, Messung der Schwierigkeit

1 Einleitung

ABP-Systeme werden heute an vielen Hochschulen zur Ausbildung in Informatik-Studiengängen entwickelt und eingesetzt [EPQ07, If15, IS01, Kr20]. Vorteile dieser Systeme sind die Messbarkeit des Programmierfortschritts, das direkte Feedback an die Studierenden und die Skalierbarkeit der Systeme, die insbesondere bei großen Kohorten wichtig wird. Die Anwendung von ABP-Systemen auf Programmieraufgaben zur Kryptologie ist ein relativ junges Anwendungsgebiet [BD15, Kn20]. Der Ansatz dabei ist, Studierende die theoretischen Grundlagen der Kryptologie (z.B. aus der Zahlentheorie) bzw. die Verwendung moderner Krypto-Bibliotheken (wie z.B. [BC]) programmieren zu lassen, um so die Berührungsängste zu nehmen und das Verständnis zu stärken („Learning by coding“). Die korrekte Implementierung wird dabei über Testfälle auf einem ABP-System überprüft. Decken die Testfälle ein breites Spektrum ab, bekommt die Lehrkraft Rückmeldung darüber, an welchen Stellen die Studierenden Schwierigkeiten haben.

Der Beitrag stellt eine Studie vor, in der 20 Studierende über ein Semester 20 Aufgaben zur Kryptologie bearbeitet haben und wertet die Ergebnisse der Tests aus. Das verwendete ABP-System erlaubt den Studierenden beliebig viele Versuche, um die vorgegebenen Testfälle zu bestehen. Die Schwierigkeit der Testfälle wird (1) über die Fehlerrate und (2) über die Bearbeitungszeit, genauer über die Anzahl der notwendigen Versuche zum

[1] Hochschule Trier, Fachbereich Informatik, Am Schneidershof, 54208 Trier, knorr@hochschule-trier.de

doi:10.18420/abp2021-5

Bestehen der Testfälle gemessen. Die Auswertung von kryptologischen Themen und Kategorien von Testfällen mittels dieser beiden Schwierigkeitsmaße erlaubt Rückschlüsse auf die Bereiche der Kryptologie, die den Studierenden besonders leicht bzw. schwerfallen. Dies ist eine wichtige Hilfe zur Verbesserung der Lehre in zukünftigen Veranstaltungen zur Kryptologie.

Die Fortsetzung des Artikels hat folgende Struktur: Kap. 2 beschreibt die notwendigen Hintergrundinformationen zum verwendeten ABP-System, zum Test Driven Software Development (TDSD) und zu JUnit-Tests. Kap. 3 erläutert den Studienaufbau, den verwendeten Datensatz und seine Auswertung. Die Ergebnisse der Studie sind Gegenstand von Kap. 4. Kap. 5 diskutiert die Ergebnisse und endet mit einem Ausblick.

2 Grundlagen

2.1 Das verwendete ABP-System ASB (Automatische Software-Bewertung)

Das System ASB wird seit 2006 an der Hochschule Trier zur automatischen Bewertung von Programmieraufgaben für Informatik-Studierende genutzt. ASB ist eine webbasierte Client-Server Applikation. Studierende und Lehrkräfte besitzen unterschiedliche Rollen und Rechte und können sich über Shibboleth anmelden. Die Daten werden in einer Datenbank gespeichert und es können Ausführungsumgebungen für verschiedene Sprachen wie Java, C++ oder Python eingebunden werden. Vorbereitend erstellen Lehrkräfte die Aufgabenstellungen und Testfälle. Die Studierenden laden ihren Programmcode zur Lösung der gestellten Aufgaben in das System. Dort werden die Testfälle auf den Code angewendet und das Ergebnis direkt im Anschluss ausgegeben. Die Studierenden haben innerhalb des Bearbeitungszeitraums beliebig viele Versuche, die Testfälle der Aufgaben zu bestehen. Weitere Informationen zu ASB finden sich in [He17].

2.2 Test Driven Software Development & JUnit Tests

TDSD stammt von Beck [Be02] und ist eine Untergruppe der agilen Software-Entwicklung. Der Grundgedanke ist nur Code zu erstellen, der vorgegebene Testfälle besteht. Entwickler können so schnell und zielgerichtet Fehler entdecken und diese beseitigen. Bei vielen ABP-Systemen erstellen die Lehrkräfte die Tests, die vom Code der Studierenden bestanden werden müssen. TDSD ist unabhängig von der Programmier-sprache. Die gängigste Java-Implementierung ist das JUnit Framework [JU]. JUnit-Tests erlauben Tests von Klassen, Methoden und Attributen. Es können auch Exceptions getestet werden, was bei vielen kryptologischen Verfahren eine wichtige Eigenschaft ist (z.B. Prüfung auf korrekte Schlüssellängen bei AES). Entwicklungsumgebungen wie Eclipse bieten graphische Oberflächen für die Visualisierung der Testergebnisse an, vgl. Abb. 1. Es gibt drei Ergebnisse eines JUnit-Tests: (1) Passed, (2) Error, (3) Failure. (1) gilt im ASB-System als bestanden (=> „true“), (2) und (3) als nicht bestanden (=> „false“).

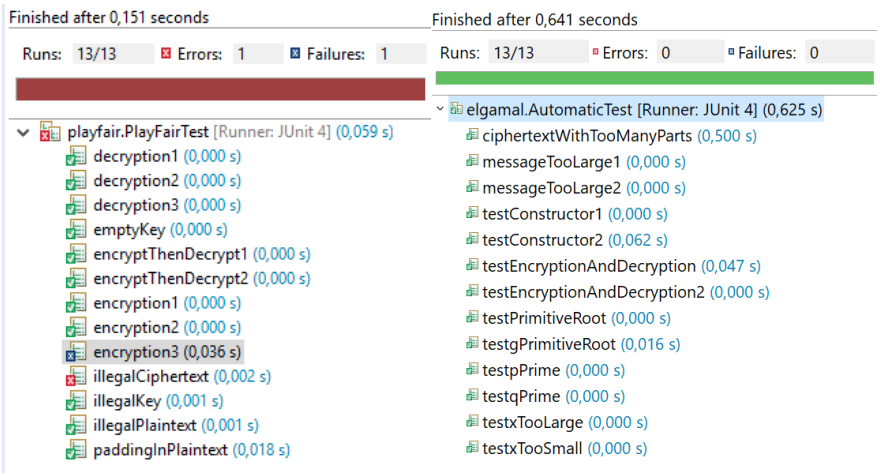

Abb. 1: Bestandene und nicht bestandene JUnit-Testfälle zur Playfair-Aufgabe (links) und zur Rabin-Aufgabe (rechts) in Eclipse

3 Studienaufbau

Das Papier basiert auf Daten des Bachelor-Wahlpflichtfachs „Kryptologisches Programmierpraktikum“ (KPP) aus dem WS 20/21. Es haben sich 20 Studierende für die Veranstaltung angemeldet, 15 haben fast alle Aufgaben bearbeitet. Die Aufgaben mussten in Java gelöst werden. Die Veranstaltung findet unregelmäßig ca. alle zwei Jahre statt, im WS 20/21 zum dritten Mal. Von den 14 Semesterwochen wurden in den ersten 10 Wochen jeweils ungefähr drei Programmieraufgaben gestellt, in den letzten Wochen bearbeiten die Studierenden ein eigenes Projekt zur Kryptologie. Die Themen der Programmieraufgaben umfassten alle gängigen Themen der Kryptologie, also u.a. symmetrische Verfahren (klassische Verfahren, Blockchiffren wie AES, Blockmodi wie CTR und GCM, Stromchiffren), asymmetrische Verfahren (RSA, Rabin, Elgamal), Verfahren zum Schlüsselaustausch (Diffie Hellman), Hash-Verfahren, Pseudozufallszahlen, X.509-Zertifikate, elliptische Kurven und ausgewählte Themen der Postquanten-Welt (NTRU, Lamport-Signaturen), vgl. [SP18]. Von den insgesamt 34 Aufgaben wurden 20 Aufgaben über ASB ausgewertet, vgl. Tab. 1. Die restlichen Aufgaben wurden händisch auf Basis des abgegebenen Codes und von dazu erstellten Videos überprüft.

Das ASB-System speicherte pro Aufgabe, Studierenden, Testfall und Zeitpunkt das Ergebnis r (für „result“) des JUnit-Testfalls. Formell:

$$r: A \times S \times T \times Z \rightarrow \{true, false\}$$

Themen	Aufgaben
Klassische Chiffren	FourSquare, Playfair, Porta (Substitutionschiffren) Jägerzaun, Skytale (Transpositionschiffren) Bifid (beides)
Asymmetrische Chiffren	BigInteger, RSA, Rabin, Elgamal, Paillier, NTRU
Signaturen	Lamport, RabinSig
Elliptische Kurven	myEC, ECDH, ECDSA
Sonstiges	Shamir Secret Sharing, Java Key Store, Padding Oracle

Tab. 1: Über ASB bewertete Themen und Aufgaben zur Kryptologie

Beispiel: *r*(Rabin, 6137, testEncryption2, 2020-10-29T13:21:35) = *true.* Dabei sind:

- *A* die Menge der Aufgaben. Über Tab. 1 kann jeder Aufgabe bei der Auswertung ein übergeordnetes Thema zugeordnet werden.
- *S* ist die Menge der Studierenden-IDs. Den Studierenden wird zur Anonymisierung eine zufällig erzeugte ID zugewiesen (im Bsp. 6137).
- *T* ist die Menge der Testfälle, typischerweise zwischen 10-20 pro Aufgabe. Die Testfälle wurden zur differenzierten Auswertung in folgende zehn Kategorien unterteilt: 1. Encryption, 2. Decryption, 3. EncDec (=encryption and subsequent decryption), 4. Signature, 5. Verify, 6. SignAndVerify (signature and subsequent signature verification), 7. Padding, 8. Constructor, 9. Illegal (Exceptions), 10. Miscellaneous. Im Beispiel: testEncryption2 ➔ Testkategorie 1, „Encryption".
- *Z* steht für die Zeit. Der Bearbeitungszeitraum für eine Aufgabe war jeweils eine Woche in der Vorlesungszeit. Die Studierenden konnten selbst wählen, wie oft und wann sie in diesem Zeitraum Lösungen ins ASB-System hochladen. Die Anzahl der Uploads schwankt zwischen 2 und 40. Für jeden Upload wurde der Zeitstempel in Sekunden gespeichert. Im Beispiel: 2020-10-29T13:21:35.

Um die Zeit zu normieren und damit vergleichbarer zu machen, wurde die Funktion *t* (für „time") verwendet:

$$t: A \times S \times T \rightarrow [0,1]$$

Dazu wurden zunächst die Testergebnisse eines Studierenden zu einem Testfall einer Aufgabe chronologisch als Liste gespeichert, bspw. [*false, false, false, true, false*]. *t* speichert den Zeitpunkt des ersten positiven Testfalls (startend von 0) geteilt durch die Gesamtzahl an Versuchen. Im Beispiel ergibt sich ein Wert von 3/5 = 0,6. Sind alle Ergebnisse „false", wird als Wert 1 gespeichert. Ist bereits der erste Versuch „true", ergibt sich als Wert 0. *t* kann daher als durchschnittliche Bearbeitungszeit interpretiert werden.

Die Fehlerrate eines Testfalls wird mit der Funktion *e* (für „error rate") gemessen:

$$e: A \times S \times T \rightarrow [0,1]$$

e teilt die Anzahl der negativen Tests durch die gesamte Anzahl an Versuchen. Beispiel: [*false, false, false, false, true*] ergibt einen Wert von 4/5=0,8. Die Funktionen *e* und *t* sind stark korreliert, erlauben aber trotzdem folgende Vergleiche bzgl. Fehlerraten und Bearbeitungszeiten:

- Vergleich innerhalb der Tests einer Aufgabe, vgl. Abb. 2.
- Vergleich zwischen Themen der Aufgaben, z.B. asymmetrischer. Chiffren vs. Signaturen oder Substitutions- vs. Transpositionschiffren, s. Tab. 1 und Abb. 3.
- Vergleich zwischen Kategorien von Testfällen, z.B. Signieren vs. Verifizieren von Signaturen oder Exception- vs. Constructor-Tests, vgl. Abb. 4.

Das ASB-System stellt die Daten in Form von Tab Separated Values-Dateien pro Aufgabe zur Verfügung. Die Dateien werden mittels Python eingelesen und ausgewertet. Die Abbildungen wurden mit Matplotlib (https://matplotlib.org) generiert. Der komplette Datensatz, die Python-Skripte sowie Beispiele für Aufgabenstellungen und JUnit-Tests stehen unter https://seafile.rlp.net/d/1776901730d14aefa869/ zum Download bereit.

4 Ergebnisse

Die Funktion *e* kann zur Auswertung der Testfälle der einzelnen Aufgaben verwendet werden. Exemplarisch wird dies an der Aufgabe zur Rabin-Verschlüsselung gezeigt, vgl. Abb. 2. Die Verschlüsselungs- und Constructor-Tests weisen deutlich niedrigere Werte für *e* und *t* auf, als die Tests zur Entschlüsselung, zur Ver- und anschließenden Entschlüsselung und zum chinesischen Restsatzes (CRT- Chinese Remainder Theorem). Dies liegt an der Definition der Rabin-Chiffre. Die Verschlüsselung ist ein einfaches modulares Quadrieren (eine Code-Zeile mit BigInteger), während die Entschlüsselung mit dem Ziehen der Quadratwurzel mod n und der Anwendung des CRT deutlich anspruchsvoller ist, vgl. [Ra79].

Abb. 3 zeigt die durchschnittliche Fehlerrate *e* pro Aufgabe. Die höchste Fehlerrate gab es bei der Aufgabe zum Padding Oracle. Diese Aufgabe war eine besonders anspruchsvolle Bonusaufgabe. Der erste bestandene Testfall lieferte hier die gesuchte Lösung. Die klassischen symmetrischen Transpositionschiffren FourSquare und Porta wurden sehr gut gelöst. Die Transpositonschiffren Jägerzaun und Skytale waren deutlich schwieriger. Die Playfair-Aufgabe war die erste im Semester, weshalb man einen gewissen Eingewöhnungseffekt berücksichtigen muss. Bei den asymmetrischen Chiffren fällt auf, dass die klassischen asymmetrischen Verschlüsselungsverfahren wie RSA, Elgamal und Rabin den Studierenden deutlich leichter fielen, als (1) die Signaturverfahren (RabinSig, Lamport) und (2) modernere asymmetrische Verfahren wie Paillier und NTRU, was evtl. auf die steigende zahlentheoretische Komplexität moderner Verfahren zurückzuführen ist. Die Aufgaben zu elliptischen Kurven liegen im Mittelfeld, was belegt, dass diese algebraische Struktur den Studierenden durchaus zugänglich ist. Bei den sonstigen Aufgaben fällt das schlechte Abschneiden der Keystore- und der BigInteger-Aufgabe auf. Java Keystores dienen dem sicheren Speichern von kryptologischen

Schlüsseln und Zertifikaten. Bei ihnen liegt die Schwierigkeit in der Komplexität der Java-Architektur dahinter, den verschiedenen Keystore-Typen und den subtilen Unterschieden zwischen diesen Typen. Die Java BigInteger-Klasse erlaubt das Arbeiten mit beliebig großen Zahlen, stellt viele wichtige Methoden für die Kryptographie zur Verfügung (wie z.B. modulare Exponentiation oder Invertierung) und eignet sich daher sehr gut als Grundlage für die Implementierung asymmetrischer Verfahren. Die Testfälle hatten keinen Bezug zu einander und wurden daher vermutlich oft chronologisch bearbeitet.

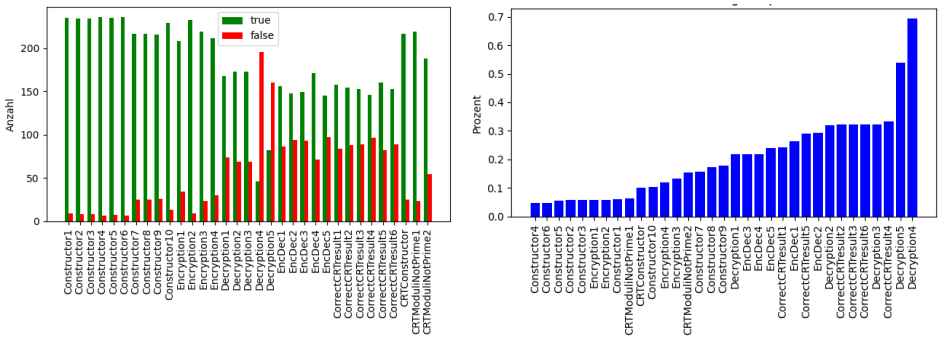

Abb. 2: Fehlerraten absolut (links) und durchschnittliche Bearbeitungszeit *t* (rechts) der Testfälle in der Aufgabe zur Rabin-Verschlüsselung

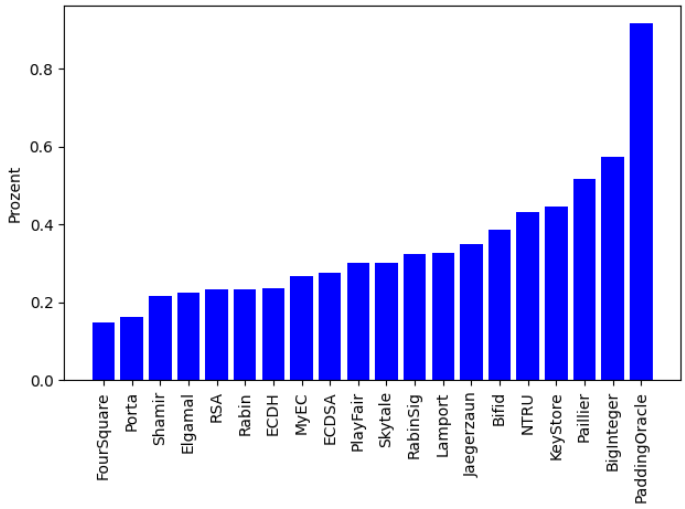

Abb. 3: Durchschnittliche Fehlerrate *e* pro Aufgabe

Abb. 4 zeigt *e* und *t* sortiert nach den zehn Testkategorien. Auffällig ist der geringe Unterschied: Tests mit hoher Fehlerrate wurden später gelöst. Tests zu den Konstruktoren und Exceptions fallen den Studierenden am leichtesten. Padding-, Encryption- und

Decryption-Tests rangieren im Mittelfeld. Die „Kombinations“-Tests EncDec und SignAndVerify schneiden schlechter ab, da hier in den Tests meist zufällige und keine fixen Testwerte verwendet wurden. Encryption fällt leichter als Decryption, Signieren leichter als die Signaturverifikation, Vertraulichkeitsschutz leichter als Integritätsschutz.

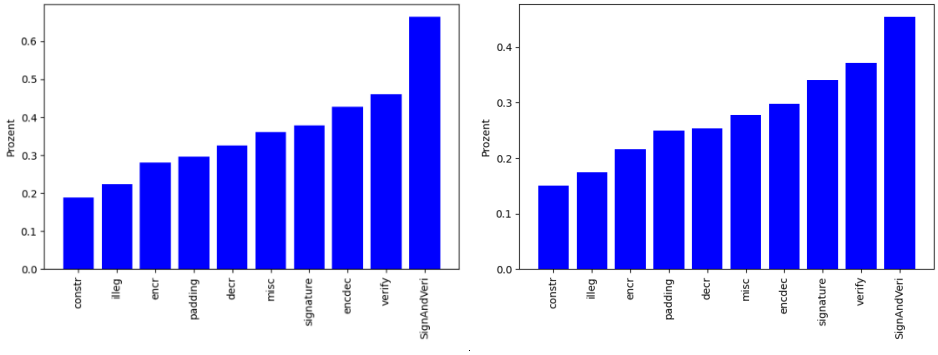

Abb. 4: *e* (links) und *t* (rechts) über alle Aufgaben gruppiert nach Testkategorie

5 Diskussion und Ausblick

Die Auswertung in Kap. 4 liefert neue Erkenntnisse bzgl. der auftretenden Schwierigkeiten bei der Programmierung von Kryptologie in Java. Lehrkräfte können so Testfälle und Aufgaben identifizieren, mit denen die Studierenden Probleme hatten und gegensteuern. Wichtige Erkenntnisse sind z.B.: Substitution fällt leichter als Transposition und Exceptions und Padding fallen leichter als Signieren und deren Verifikation

Bei der Studie sind allerdings folgende störenden Einflüsse zu nennen. **Unterschiedliche Voraussetzung bei den Studierenden**: Die Fehlerrate bei den Testfällen wird durch die Vorkenntnisse der Studierenden in der Kryptologie und der Java-Programmierung beeinflusst. Viele Studierende hatten zu Beginn noch keine intensiven Erfahrungen in Java und steigerten diese im Verlauf der Veranstaltung. Eine gewisse Affinität zu Themen der Kryptologie und Java war bei allen Studierenden gegeben. **Plagiate**: Ein Plagiatsschutz ist zwar im ASB-System vorgesehen, war aber für KPP nicht aktiviert. Plagiate mussten im Verdachtsfall manuell geprüft werden. Der Source Code jeder Abgabe ist dafür in ASB vorhanden. Da viele Aufgaben zeitaufwändig waren und Einfluss auf die Note hatten, kann ein gewisses Maß an „Teamarbeit“ zwischen den Studierenden nicht ausgeschlossen werden. **Chronologie der Testfälle**: ASB präsentiert die Testfälle pro Aufgabe in einer fixen Reihenfolge. Einige Studierende versuchten zunächst, die Testfälle genau in dieser Folge abzuarbeiten.

Effenberger et al. [ECP19] unterscheiden zwischen „Difficulty“ und „Complexity“ von Programmieraufgaben. Für die Difficulty werden wie hier Fehlerrate und Bearbeitungszeit

empfohlen. Als Komplexitätsmaß werden u.a. die Anzahl der Codezeilen und der Kontrollflussstrukturen genannt, die auch hier zum Einsatz kommen könnten. Damit ließen sich die Probleme bei der Lösung der Aufgaben noch genauer analysieren.

Für die Zukunft von KPP sind weitere Aufgaben und eine bessere Kategorisierung der Testfälle und Aufgaben geplant. Bisher sind die Kategorien und Themen in einigen Fällen nicht disjunkt. Bei den ASB-Aufgaben überwiegen die asymmetrischen Verfahren. Auch Aufgaben zu modernen symmetrischen Verfahren sollen automatisiert in ASB überprüft werden. In ASB bietet sich eine Erweiterung um eine Auswertung gemäß Abb. 2 an. Außerdem könnten die Aufgaben auch in Python gestellt werden. Dies würde einen Vergleich der Programmiersprachen Java und Python erlauben. Vorarbeiten dazu laufen [Me21].

Literaturverzeichnis

[BC] Bouncy Castle Homepage, https://www.bouncycastle.org, letzter Zugriff am 2.6.2021.

[BD15] Braga, A., Dahab, R.: A Survey on Tools and Techniques for the Programming and Verification of Secure Cryptographic Software, Proceedings of XV SBSeg, 2015.

[Be02] Beck, K.: Test-Driven Development: By Example. Addison Wesley, 2002.

[ECP19] Effenberger, T., Čechák, J., Pelánek, R.: Measuring Difficulty of Introductory Programming Tasks. Proc. of the 6th ACM Conf. on Learning@Scale, pp. 1-4, 2019.

[EPQ07] Edwards, S., Pérez-Quiñones, M.: Experiences using test-driven development with an automated grader. Journal of Computing Sciences in Colleges 22(3), pp. 44-50, 2007.

[He17] Herres, B., Oechsle, R., Schuster, D.: Der Grader ASB. In: "Automatisierte Bewertung in der Programmierausbildung", Waxmann-Verlag, S. 255-271, 2017.

[If15] Iffländer, L. et al.: PABS – a Programming Assignment Feedback System. In: Proc. of the 2. Workshop Automatische Bewertung von Programmieraufgaben, 2015.

[IS01] Isong, J.: Developing an automated program checker. J. Computing in Small Colleges, 16, 3, pp. 218-224, 2001.

[JU] JUnit Homepage, https://junit.org, letzter Zugriff am 7.6.2021.

[Kn20] Knorr, K.: Learning and Grading Cryptology via Automated Test Driven Software Development. In: Proc. of the 13th IFIP WG 11.8 World Conference on Information Security Education (WISE), WISE 13, Maribor, Slovenia, pp. 3-17, 2020.

[Kr20] Krugel, J. et al.: Automated Measurement of Competencies and Generation of Feedback in Object-Oriented Programming Courses. In: 2020 IEEE Global Engineering Education Conference, EDUCON 2020, Porto, Portugal, April 27-30, S. 329-338, 2020.

[Me21] Meiser, J.: Automatische Bewertung von kryptologischen Programmieraufgaben in Python. Projektarbeit Informatik, Hochschule Trier, 2021.

[Ra79] Rabin, M.: Digitalized Signatures and Public-Key Functions as Intractable as Factorization. MIT Laboratory for Computer Science, 1979.

[SP18] Stinson, D., Paterson, M.: Cryptography: Theory and Practice. 4th edn. CRC, 2018.