Einführung in das wissenschaftliche Arbeiten
WS 2007/2008 am IAIK

Roderick Bloem

http://www.iaik.tugraz.at
Assignment D1

Flavius Aspra
Georg Doppler
Genc Fejza
✓ Christian Gamperl
✓ Marco Gruebner
✓ Olga Karpovych
✓ Stephan Peijnik
✓ Martin Prinz
Natalie Solderer
Kevin Stai
✓ Phillipp Url
Typical Paper Structure

- **Title**
  - You paper in a few words.
  - catchy
  - specific enough to say what you do
  - broad enough to “claim” an area

- **Authors**
  - including institution

- **Abstract**
  - Your paper in a few sentences

- **Introduction**
  - Seen next slide

- **Preliminaries**
  - For theoretical papers: mathematical background, definitions

- **Contribution**

- **Experimental Results**
  - Whether or not your idea works should be separated from the idea itself

- **Related work**
  - sometimes here, sometimes part of introduction

- **Conclusions**
  - Summary again

- **References**
# Structure of Introduction

<table>
<thead>
<tr>
<th>What?</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background</strong>: what are we talking about?</td>
<td>Formal specifications enjoy increasing importance… . A specification is unrealizable if …</td>
</tr>
<tr>
<td><strong>Motivation</strong>: why is there a problem?</td>
<td>Debugging a formal specification is more difficult than debugging software…</td>
</tr>
<tr>
<td><strong>Summary</strong>: what is our solution and why is it good?</td>
<td>In this paper we present an approach for debugging unrealizable specifications based on the following idea:…</td>
</tr>
<tr>
<td><strong>Related work</strong>: What have others done &amp; why is our work different?</td>
<td>The idea of … was previously suggested in … in the context of …. Our work focuses on …. Cimatti et al. [6] proposed to …; we discussed the improvements over their work above. We are not aware of any previous work on ….</td>
</tr>
<tr>
<td><strong>Flow of paper</strong></td>
<td>The rest of the paper is organized as follows. Section 2 will…</td>
</tr>
</tbody>
</table>
Paragraphs

• Paragraphs reflect structure: one idea per paragraph.
• This helps the reader understand when you start a new thought
• For the introduction, you may find one paragraph for each of the point listed on last slide. Sometimes more, if a point is complicated.
• Try to group related sentences together, not spread them across the paper
A Semantic Approach to Ornithology

B. Watcher
Massachusetts Institute of Taxonomy

Abstract. The traditional approach to ornithology is empirical. We argue that this approach has its shortcomings and that a semantic approach is preferred. We support our thesis with a semantic study of passer domesticus.

1. Introduction

The first paragraph typically introduces the background of the paper. In this case, we will state that we talk about ornithology, or rather about the theory of ornithology.

In the second paragraph, we introduce the motivation. That is, we explain that something is wrong with the state of affairs. Ornithological theory is not complete without a semantic approach, whatever that may be. Of course, the real reason for writing the paper is that the author wants a PhD, but we do not mention that here.

In the third paragraph, we come to speak of the actual contribution. We explain what a semantic
First Sentence

A good first sentence introduces the topic or summarizes the paper.

Examples are taken from reports.
1. “Generally, scientific papers have the same structure.”
2. “The whole paper should be separated into five bigger sections:…”
3. “The purpose of this report is to highlight the differences between four scientific reports and draw conclusions from them”
4. “I have skimmed over all four reports that you have given us to review for this assignment to get the general idea about them, instead of reading all of them.”
Hints for Reports

• Find a good title
• Use only one language
• Put care into the structure: sections, paragraphs, sentences
• Have an introduction and a conclusion. (Perhaps a section, perhaps just a sentence)
• Avoid “seasaw reasoning”:
  – Bad: “It may be fun to go out during the rain. However, you may get wet. But umbrellas help against that. Of course, you may still get wet even though you have an umbrella. (pro-con-pro-con)
  – Better: Even though umbrellas are not perfect, they provide good protection against the rain. Thus, there is no reason not to go out and have fun when it rains!
  – It took me a few iterations to get this sentence right, including finding out what I wanted to say: go out or not go out?

Professor Horst Cerjak, 19.12.2005
Hints for Reports

• Be careful with opinions: be sure to back them up
• Be careful with words like “never,” “always,” etc. Is your statement really true?
• Remember: your reader is skeptical.

• Get someone to proofread you report. Have her check for structure, ease of reading, English
  – Don’t use colloquial English. “Well, you can’t have it all.”
  – Don’t overpolish your English. “Precipitation (especially when liquid) has the tendency to leave one doused.” → “Rain makes you wet.”
Phrases

Find the structure of the argument in the next slide. (Paragraph breaks removed by me.)

We present a critical evaluation of the first known implementation of elliptic curve cryptography over \(F_{2^p}\) for sensor networks based on the 8-bit, 7.3828-MHz MICA2 mote. We offer, along the way, a primer for those interested in the field of cryptography for sensor networks. We discuss, in particular, the decisions underlying our design and alternatives thereto. And we elaborate on the methodologies underlying our evaluation. Through instrumentation of UC Berkeley’s TinySec module, we argue that, although symmetric cryptography has been tractable in this domain for some time, there has remained a need, unfulfilled until recently, for an efficient, secure mechanism for distribution of secret keys among nodes. Although public-key infrastructure has been thought impractical, we show, through analysis of our original implementation for TinyOS of point multiplication on elliptic curves, that public-key infrastructure is indeed viable for TinySec keys’ distribution, even on the MICA2. We demonstrate that public keys can be generated within 34 seconds and that shared secrets can be distributed among nodes in a sensor network within the same time, using just over 1 kilobyte of SRAM and 34 kilobytes of ROM. We demonstrate that communication costs are minimal, with only 2 packets required for transmission of a public key among nodes. We make available all of our source code for other researchers to download and use. And we discuss recent results based on our work that corroborate and improve upon our conclusions.
We present a critical evaluation of the first known implementation of elliptic curve cryptography over $\mathbb{F}_{2^p}$ for sensor networks based on the 8-bit, 7.3828-MHz MICA2 mote. We offer, along the way, a primer for those interested in the field of cryptography for sensor networks. We discuss, in particular, the decisions underlying our design and alternatives thereto. And we elaborate on the methodologies underlying our evaluation. Through instrumentation of UC Berkeley’s TinySec module, we argue that, although symmetric cryptography has been tractable in this domain for some time, there has remained a need, unfulfilled until recently, for an efficient, secure mechanism for distribution of secret keys among nodes. Although public-key infrastructure has been thought impractical, we show, through analysis of our original implementation for TinyOS of point multiplication on elliptic curves, that public-key infrastructure is indeed viable for TinySec keys’ distribution, even on the MICA2. We demonstrate that public keys can be generated within 34 seconds and that shared secrets can be distributed among nodes in a sensor network within the same time, using just over 1 kilobyte of SRAM and 34 kilobytes of ROM. We demonstrate that communication costs are minimal, with only 2 packets required for transmission of a public key among nodes. We make available all of our source code for other researchers to download and use. And we discuss recent results based on our work that corroborate and improve upon our conclusions.
Science?

Find a problem to solve
- read papers? Papers propose solutions, not problems!
- Problems must fit your skills
- Finding problems is a PhD student’s nightmare

Find a solution
- Solution typically does not fit problem – change problem
- If you find one solution, you probably find ten
- Each solution begets ten problems
- Solutions need to be tried – need students!

Get money to fund students

Write paper
- Find consistent story that fits chaotic process
- Only way to get your thoughts straight
- Only way to find holes, new problems, other solutions, extensions
- Research begins when writing the paper?

Find time for research in between writing proposals

Learn state of the art
- No time? Teach it!
Philosophy of Science

Karl Popper
• Falsification: Hypothesis and test

Thomas Kuhn
• Paradigm shifts
Topics

1. What is the economic damage due to bugs?
2. Societal aspects of bugs – security, privacy, etc.
3. Which psychological factors influence how good written code is?
4. Legal aspects of bugs – who is liable?
5. Bugs in cars – why is software so much harder than mechanics?
6. How does a debugger work?
   – Stefan Peijnik
7. How does a bug tracking system work?
8. Redundant computing: how and why
   – Christian Gamperl
10. When you know you have a bug, how do you find where it is?
11. How should programs deal with bugs when they occur?
12. How do you create good software tests?
   – Philip Url
13. Test automation
14. Why is it hard to write good software specifications?
15. The evolution of programming languages in the pursuit of program correctness.
   – Marco Gruebler
16. The evolution of good programming styles in the pursuit of program correctness (coding guidelines).
17. How does Java help avoiding bugs?
18. Software development models and correctness
   – Martin Prinz
19. How to avoid bugs – the programmer's view
20. How to avoid bugs – the manager's view
21. How do people avoid bugs in high-security applications?
22. Why can’t you just prove correctness of a program?