Teaching Statement

Andrew Miller

November 2015

My teaching philosophy is about building a harmony between formal rigor and intuitive understanding.

For me, as a primarily intuitive learner, I feel I only really understand a concept once I build a internal model that I can apply in creative problem solving. In my natural teaching style, I often make use of animations and whiteboard illustrations, narrate example scenarios, and ask questions to identify the mental models held by my students.

For the many formal skills needed in security-related topics, such as cryptography and software verification, I find that programming projects are what have helped me bridge the gap. Therefore in my teaching I like to use carefully-designed programming challenges, ideally ones that match as closely as possible the mathematical notation used on-paper.

Teaching Cybersecurity with Cryptocurrencies. I have enjoyed several opportunities to teach collaboratively and develop educational materials. These experiences have centered around a common topic (cryptocurrencies), but have spanned a variety of teaching contexts. These include contributing lectures to the Princeton online course and textbook on cryptocurrency technologies, developing project assignments for undergraduate courses at UMD, and co-instructing a cryptocurrency research seminar course.

With my co-advisor Prof. Elaine Shi, I designed and coordinated a “smart contracts” lab for an undergraduate security course at Maryland (CMSC 416). Briefly put, smart contracts are programs that run on top of a cryptocurrency platform. These programs are trusted to execute correctly and to stay available; however, inputs to these programs are “public” and immediately leaked to any attacker. This turns out to be a convenient pedagogical tool, for two main reasons. First, since the platform does not inherently guarantee privacy, it is necessary to apply cryptography techniques to implement many tasks. Second, the use of virtual currency makes the adversarial model natural and clear.

These insights are best explained by example: an elementary smart contract application is a “Rock-Paper-Scissors” game. Two parties each deposit $1 of virtual currency into the smart contract instance representing this game. Next, each player picks a move (rock, paper, or scissors), and finally the winner receives $2. A naïve implementation of this game as a smart contract may easily encounter a pitfall, where the first player’s choice is leaked in plaintext, allowing the second player to always win. This hazard provides a natural motivating example to teach hash-based commitments: the problem is solved by having both players initially commit to their move, and then only after both commitments are recorded published are they opened.

We found through this course project that a wide range of essential cryptographic techniques can be similarly motivated and illustrated by smart contract applications. We also found that we could appropriate tools developed by the open source community. In particular, we used Pyethereum, an

https://www.coursera.org/course/bitcointech
existing cryptocurrency implementation that features a “smart contracts” simulator and program interpreter. After the course, we distilled our expanded examples, programming guides, and development environment, into a self-contained suite of course materials, which is now freely available[^2].

My experience was similar developing a project for an undergraduate networking course, (CMSC 417).[^3] For this project, the task is to implement a peer-to-peer node. I was able to adapt an existing implementation of the Bitcoin “message parsing” library to provide as a project skeleton. Appropriating existing tools this way sidesteps many tedious and error-prone details that are irrelevant to the course curriculum, and instead allows students to focus on the desired curriculum of robust socket handling and the protocol logic.

Courses I would like to Teach. I can teach a wide variety of classes ranging from theory, to programming languages to systems, and especially the following:

- A broad computer security course for undergraduates, including topics and techniques from cryptography, distributed systems, network security, and software security. I would use cryptocurrency applications as a running example threaded throughout much of the material.
- An interdisciplinary graduate seminar course, specifically scoped for cryptocurrency as a topic. This would involve applying techniques from cryptography, systems security, and programming languages.
- Programming languages, especially including type systems for functional programming, and program analysis. I would especially prefer to focus on topics and language abstractions related to concurrency.
- Introductory programming, algorithms, networking, or cryptography. In these classes, I would want to include an emphasis on software security.

References


[^2]: [http://mc2-umd.github.io/ethereumlab](http://mc2-umd.github.io/ethereumlab)