# Developing an Applied, Security-Oriented Computing Curriculum


## Abstract

Software and hardware security is a reality that all stakeholders must face, from hardware engineers to software developers to customers. As a direct result, the technology industry is facing a growing need for engineers who understand security principles at varying levels of abstraction. These engineers will need security-oriented perspectives stemming from both theoretical and practical disciplines, including software engineering, computer engineering, and computer science. Unfortunately, in traditional academic settings, secure software and hardware are typically taught independently despite being intertwined in practice. Consequently, the objective of this initiative is to prepare students to apply a security-oriented awareness to a broad range of hardware and software systems by developing a multi-disciplinary curriculum involving three departments. Our efforts at Rochester Institute of Technology focus on integrating security into software design and implementations, hardware design and implementations, and hardware-software co-design. In the cluster of courses described in this paper, we use cryptographic applications as the motivating security focus. We describe changes made to an existing introductory cryptography course, report on a recently-developed course entitled Hardware and Software Design for Cryptographic Applications, and present our plans for a Secure Software Engineering course.


## 1. Introduction

With the pervasiveness and importance of cyberinfrastructure in modern society, secure computing and communication have become critically important. Applications with important security requirements include e-commerce, voice/video communications, military operations, secure databases, and financial market transactions. As a result, the technology industry has a growing need for secure infrastructure at lower levels, such as disk drives, processors (e.g., built-in encryption/decryption), trusted system boards, network switching elements, mobile devices, and sensors. Security at any point in the infrastructure requires careful attention to algorithm choice and implementation method, with trade-offs between software and hardware being particularly important at these lower levels. The development of these secure systems requires a population of entry-level developers who have knowledge and skills beyond standard hardware and software design. A key element of this is an understanding of cryptographic algorithms and their implementations. To gain the necessary knowledge and skills, a student must learn concepts from multiple disciplines including computer engineering, computer science, and software engineering.

In traditional computing curricula, computer engineering students usually learn how to design general purpose digital systems, but they lack the knowledge related to the design of specialized cryptographic circuits and optimizations of hardware-software co-designs. Those students who do some cryptographic design might construct a hardware implementation of the algorithm without knowing the fundamental theory on which it is based. Computer science and software engineering students study cryptographic algorithms mostly as a rigorous mathematical exercise with some software implementation. They rarely investigate the performance of their implementations and are usually not familiar with software optimizations or hardware implementations. In many applications, implementation aspects are crucial because

cryptographic algorithms have demanding resource requirements. This is especially true when they operate upon data from a streaming communications channel, with a constrained power budget, or in an environment where side-channel attacks can be exploited to compromise the system.

To study security across multiple disciplines - computer engineering, software engineering, and computer science – we are developing a cluster of applied cryptography courses, and created a laboratory with state-of-the-art field programmable gate array (FPGA) hardware boards and development stations tailored for the study of efficient software, hardware, and combined hardware-software implementations. The goal of the Hardware and Software Design for Cryptographic Applications course is to build knowledge and skills necessary for efficient and secure implementations of cryptographic primitives in software and hardware. The goal of the Engineering Secure Software course is to teach students how to apply security principles to every phase of the software development lifecycle.

We are confident that students who take the courses that this project develops will be aware of varied approaches for implementing cryptographic algorithms along with techniques to assess their strengths and weaknesses, and measure system performance. They will also be able to apply best practices to design a broad range of secure software systems. Finally, these courses will increase awareness and interest in cryptography and secure computing which is needed for an overall improvement in system security.

## 2. Role of Cryptography in Multidisciplinary Computing Education

Cryptographic algorithms, and their secure implementations, are required in a growing number of computing systems. The specification, selection, and development of cryptographic solutions require consideration of algorithm performance, software design, and hardware design. The current study of cryptography typically does not cross disciplines to span this range of skills from fundamental theory to practical implementation.

This new curriculum introduces students across multiple computing disciplines to the broad range of topics associated with cryptographic algorithms and their realization in hardware and software. Students engage in multi-disciplinary activities on real cryptographic problems, extending their knowledge and skills well beyond that of their home discipline. Students who possess this broad range of knowledge and skills will be more effective developers of secure systems.

## 3. Cryptography Courses at Rochester Institute of Technology

For several years, we have been offering coursework in cryptography in the Department of Computer Science. Cryptography I, a traditional introductory course, covers block ciphers, hash functions, and public-key systems together with the mathematics behind it. In the last offering of Cryptography I, we introduced a new thread on efficiency and secure implementations of cryptographic algorithms. This new material encourages students to enroll in one or both of the new courses discussed below.

## Cryptography I – Introduction to Cryptography

In the basic cryptography course offered in Computer Science, we attempt to provide students with a balanced mathematical approach but avoid very formal concepts of security. This course includes the topics typically covered in an undergraduate cryptography course, such as, the private- and public-key cryptosystems with mathematical background, or authentication methods. We do, however, spend significant effort on studying the computational side of various cryptographic primitives and protocols. Students complete programming assignments of a limited scope, but typically they do not involve fine-tuned implementations needed by real applications. In addition, hardware issues shaping many of the current requirements of industrial cryptographic protocols are only mentioned in passing. The main new component of the modified Cryptography I course makes students more sensitive to efficiency and practical limits of cryptographic applications. First, the students learn about feasibility of cryptographic algorithms (or desired infeasibility of attacks on them) from the point of view of the general theory of algorithms and computational complexity theory. What, in principle, could work, and what rather will not. The redesigned Cryptography I course also includes basic hardware considerations in the design of heavily used primitives, for example in the Advanced Encryption Standard (AES) or in the recently launched NIST SHA-3 competition for the design of a new hash standard. The discussion of side-channel attacks was added to the basic course, though more technical study of the feasibility of such attacks is delayed to a following course.

## Cryptography II – Advanced Cryptographic Algorithms

This course investigates advanced topics in cryptography. It begins with an overview of the necessary background in algebra and number theory, private- and public-key cryptosystems, and basic signature schemes. The course covers the number theory and basic theory of Galois fields used in cryptography; history of primality algorithms and the polynomial-time test of primality; discrete logarithm-based cryptosystems including those based on elliptic curves; interactive protocols including the role of zero-knowledge proofs in authentication; construction of untraceable electronic cash on the net; and the basics of quantum cryptography. Other topics may include digital watermarking, fingerprinting, and steganography. Programming exercises are required.

## 4. Hardware and Software Design for Cryptographic Applications

The primary goal of this course is to build the knowledge and skills necessary for efficient and secure implementations of cryptographic primitives in software and hardware. The implementation platform is a field programmable gate array (FPGA) containing a general purpose processor and additional reconfigurable fabric for implementations of custom hardware accelerators. Student teams design selected cryptographic algorithms then compare and contrast various implementation alternatives, such as software, custom FPGA hardware, and hybrid hardware-software co-design. Project teams are ideally composed of one computer engineering student and one software engineering or computer science student.

## Structure

A list of topics covered in the course is shown in Table 1.

Table 1 - Hardware-software design course topics

| Lecture topic | Hands-on assignments |
|---|---|
| Security need, applications of private- and public-key cryptosystems, overview of public-key cryptosystems, cryptographic hash functions and their applications | |
| General hardware-software co-design flow for platform FPGAs, hard- and soft-core processors, communication interfaces, performance, flexibility and cost trade-offs | Using Xilinx Platform Studio (XPS) to create hardware and software for an FPGA-based system-on-chip;<br>Profiling software with hardware timers, interrupts, advanced software design flow with Xilinx Software Development Kit (SDK) |
| FPGA technology overview, binary finite field arithmetic, block ciphers, Advanced Encryption Standard | Implementing the AES-128 block cipher in software, profiling software with hardware timers and gprof, modifying code to improve performance, performance gain/cost analysis |
| Hardware-software co-design with Impulse C | Impulse C tutorial, using Impulse C to port the AES algorithm to run on the FPGA board, performance gain/hardware cost analysis, exploiting parallelism and pipelining at the software level, coding techniques for parallelism and pipelining |
| Block cipher modes: ECB, CBC, OFB, AES-Galois/Counter Mode | AES-Galois/Counter Mode software implementation |
| Introduction to design of custom hardware accelerators, VHDL modeling for synthesis, overview of communication interfaces, design flow and hardware-software debugging | FPGA design flow with the Xilinx toolchain and VHDL |
| Area and performance oriented hardware architectures of AES, overview of secure hardware design techniques | Implementing custom AES hardware component |

**Material**

At a high level there are four different areas of material that are covered in both lecture topics and lab assignments: 1) cryptographic foundations, 2) FPGA-based embedded systems, 3) embedded software development and optimization, and 4) hardware and hardware-software design and development. These different areas were unified through the various stages of development of the AES block cipher and SHA-3 hash functions[1,2,3].

1) Cryptographic Foundations

Although AES is the primary candidate for lectures and exercise material due to its prominence as the leading block cipher, other fundamental cryptographic primitives were discussed to enlighten students about the ubiquitous need for security. In particular, topics on hash functions and the SHA-3 competition, RSA, ElGamal, and elliptic curve public-key cryptosystems[4] were integrated into the course schedule. Basic mathematical properties and operations that are utilized in such designs were covered in order for the students to grasp the specific details of these algorithms.

2) FPGA-Based Embedded Systems

Due to a strong interdependence of hardware and software in FPGA-based embedded systems, as well as the numerous performance, power, and size constraints that these systems face, we felt it was appropriate to use them as the basis for all of our development work in this course. In particular, this choice allowed us to look at:

- Algorithm acceleration with spatial (parallel) computing in an FPGA fabric.
- Flexibility of processor-based computing using high-level software tools.
- Exploiting tightly coupled hardware and software domains for performance and flexibility benefits, though only addressing some of the practical aspects. A more in depth treatment was described by Schaumont[5].

In this course, we utilized the Xilinx ML507, a general FPGA-based development platform. Lectures addressed the underlying FPGA technology specific to this platform, including details related to the FPGA fabric and configurable logic blocks, system buses and memory interfaces, and hard-core PowerPC and soft-core MicroBlaze processors.

3) Embedded Software Development and Optimization

Since one of the threads of the course was focused on translating cryptographic algorithms and designs into running software and hardware, a fundamental lecture and lab topic was profiling and analyzing these implementations to identify areas for potential performance improvement. Student optimization efforts in these areas were mainly focused on the profiling techniques used in embedded software environments and the different source code optimization techniques that can improve performance in localized areas of the software. Some instrumentation, profiling, and optimization techniques that the students used were:

- Using timer components as an instrumentation mechanism in an existing embedded system to profile an application using raw cycle count measurements.
- Using the GNU profiling tool gprof to analyze source code.
- Experimenting with common source code optimization techniques, such as data width adjustments, static versus dynamic memory allocation, variable scope relocation, loop manipulation, and function invocation.

4) Hardware and Hardware-Software Design and Development

To facilitate the initial hardware and hardware-software development we chose to use Impulse C, a C-to-FPGA programming model and compilation engine that promotes rapid prototyping of digital hardware using a subset of the C programming language. Using Impulse C as an efficient tool in development of hardware systems by software oriented students was studied by Dandass[6]. In our course students used it to transform their existing software designs for AES to hybrid hardware-software solutions that performed at significantly higher levels.

Once the students finished experimenting with their Impulse C designs, their next and final task was to take their existing designs one step further and focus on custom hardware accelerators for the AES encryption engine. This involved an understanding of basic hardware modeling and digital design concepts. The assignments in this portion of the course were led by the computer engineering students. To account for students without an extensive hardware background, a single class was dedicated to an overview of VHDL syntax, development practices, and modeling techniques.

Once the students became acquainted with hardware modeling techniques and design flows, we provided them with a working implementation of the AES engine. After verifying the functionality of this model using a provided test bench, the students were asked to modify this implementation to shorten the width of the data path using a folded register architecture. The overall workflow for this portion of the course is shown in Figure 1.

The first stage depicts the original software version of AES as it was run on the ML507 development board. The top two stages show the incremental development cycle we used to translate the software version into a hardware-software co-designed application using Impulse C. The bottom stage depicts the design of the custom hardware accelerated version of AES, where the software was only responsible for driving the internal logic of the hardware component.

**Project**

The SHA-3 hash function candidates were the basis for the course project due to their current and future widespread influence in academia and industry. With the final round of the NIST SHA-3 competition in place, research efforts have been increasingly focused towards the five remaining candidate functions: BLAKE, Grøstl, JH, Keccak and Skein[3]. Given the importance of selecting the candidate that provides the best combination of security, performance, simplicity, and modularity, many teams across the world have focused on the analysis of the mathematical and statistical properties, and the implementation efficiency of these five hash functions.

In our course, we divided the students into five teams, each focusing on a single SHA-3 finalist. The term-long project had four major parts culminating with a research paper and presentation by each team. The highlights of each part were:

- Researching the history of hash functions, the current standard hash functions, and recent advancements in cryptanalysis efforts that target hash functions.

- Discussing the internal algorithmic details of the team's hash function, which serves as a basis for the remaining parts of the project.

- Analyzing published research efforts on the design, implementation and performance of the hash function.

- Implementing the hash function in software using the publicly available resources provided by each SHA-3 candidate team.

In order for students to have a common baseline for comparison of their results, all teams implemented the 512-bit digest versions of these hash functions. All teams quantified their implementation's performance by collecting cycles/byte measurements on fixed sized messages.

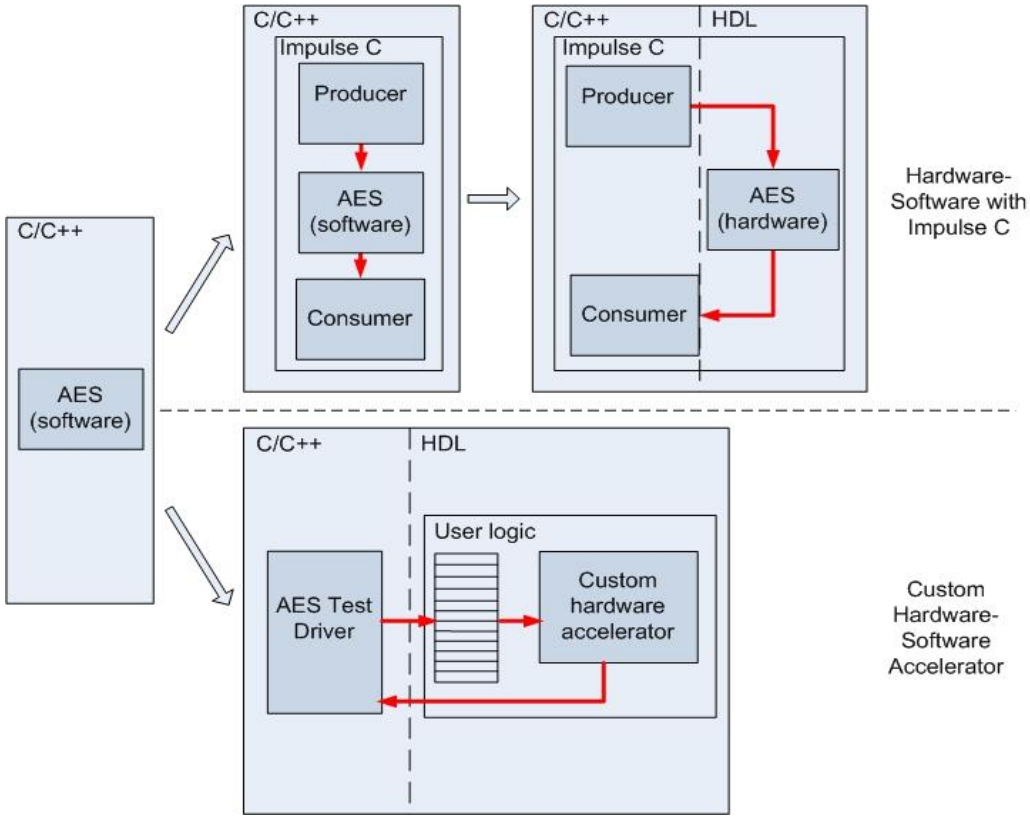


Figure 1 - Incremental development of AES as studied in the course.

## Outcomes

The overall objective of this course is to equip future engineers with the knowledge and skills necessary to implement security-related applications, specifically, cryptographic algorithms, across both hardware and software. We designed the course to give students the ability to:

- Customize and implement an FPGA based embedded processor system in order to build and run cryptographic primitives.
- Profile software applications and identify performance bottlenecks.
- Apply optimization techniques to improve application performance at different levels of abstraction.
- Analyze performance costs and gains in both hardware and software.
- Use Impulse C to perform high-level synthesis of C programs into FPGA hardware.
- Implement a custom hardware accelerator for the selected architecture to achieve performance and area goals.

**5. Engineering Secure Software**

In the spring term of 2012, we will be introducing a new undergraduate course developed in the Department of Software Engineering entitled Engineering Secure Software. The primary goal of this course is to equip students with the skills, principles, and knowledge needed to develop secure software. While related courses have focused on defensive coding practices[7,8,9], information assurance[10], and software testing[11], our course will have the added emphasis of software engineering practices such as requirements engineering, secure designs, maintenance, and assessment. The course will start as an upper-class elective seminar, but will be required for all undergraduate software engineering majors starting in fall 2012.

**Structure**

The class will meet twice per week over nine weeks with two-hour sessions in a combined lecture-laboratory setting. The schedule for each day is a 10-minute Vulnerability of the Day (VotD), a 40-minute lecture of the day's material, and an hour-long associated activity.

The purpose of the VotD is to cover concrete, code-level security vulnerabilities found in production-level software. In an effort to maintain student interest, we chose to break up the discussion of vulnerabilities as one VotD each class session, which eliminates the monotony of learning many different vulnerabilities in one or two sessions. This also provides the students with both abstract principles and concrete code examples each day in class. Many of the vulnerabilities come from the Common Weakness Enumeration (CWE), with an emphasis on the Top 25 Vulnerabilities[12]. Examples of the VotD include: SQL injection, cross-site scripting, buffer overflow, and HTTP response splitting. Each VotD is taught as a potential coding mistake that a developer can make. In each VotD, the instructor covers a code example, the potential threat of the vulnerability, and mitigations.

A lecture follows after the VotD. The purpose of the lecture is to cover the theoretical and practical principles behind secure software as it applies to the day's phase of the software development lifecycle (SDLC). For example, in covering the design phase, we cover security risks associated with conventional design patterns, along with a collection secure design

patterns[13]. After the lecture, the material is reinforced by a daily activity. Working in pairs, students will follow tutorials on a tool or technique. Some days are devoted to learning and applying a given software tool, other days involve an interactive exercise with teammates.

Table 2 - Mapping security principles to software engineering practices

| Software engineering practice | Topics covered | Tools and activities |
|---|---|---|
| Requirements | Abuse cases, personas, anti-requirements, compliance, privacy | |
| Planning | Risk assessment, threat modeling | Microsoft Threat Modeling tool[15] |
| Design & Modeling | Secure design patterns, architectural risks, role-based access control, formal model checking | Alloy[16] |
| Implementation | Defensive coding practices, code reviews, static analysis | Vulnerability of the Day, web application activity |
| Testing | Unit test generation, fuzz testing, penetration testing, exploratory testing | JTest[17] |
| Deployment | Deployment of cryptographic algorithms and networks, sandboxing, authentication | Wireshark[18], Java Security Manager |
| Maintenance & Assessment | Common Vulnerability Scoring System[19], patch management, responsible disclosure | |

**Material**

With the target audience being software developers, the emphasis of the course is on secure software over security software. Our guiding principle is to show how security applies to each practice of the SDLC[14]. Practices of the SDLC include: requirements, planning, design, modeling, implementation, testing, deployment, maintenance, and assessment. A map from software engineering practice to the security topics covered can be found in Table 2.

One may notice that "classic" security practices, such as networking and cryptography, are covered, yet with the focus on software deployment. With our target audience being developers,

the emphasis is on providing the knowledge they need to select and use cryptographic software appropriately.

**Projects**

The course contains two major projects: a project case study and a programming assignment.

For the case study, students will work in teams of two or three to assess the security of a large, open source software product. Students are required to select a case study that has (a) significant security implications if exploited, (b) a public record of fixed vulnerabilities, and (c) publicly-available source code. The project involves incrementally submitting three parts of a paper: domain analysis, design analysis, and code analysis. As the material is covered in class, students analyze their own case studies to enumerate the threats to their case study's domain, potential design-level vulnerabilities, as well as potential code-level vulnerabilities. Teams who find and report previously unknown security vulnerabilities in their case study will receive extra credit. To aid the open source community, the best papers will be disseminated to the development teams associated with the software products analyzed in the papers.

In addition to the case study, students will work in teams to build a custom web application fuzz tester (i.e. "fuzzer"). A web application fuzzer is a program that searches a website for its inputs and automatically constructs potential exploits for those inputs. The result is a sequence of HTTP requests sent to the system under test and resulting responses that the fuzzer processes in an effort to find a vulnerability.

While web application fuzzing tools exist, using them in practice requires a significant amount of customization effort. Given a simple skeleton program, students will develop a fuzzer that can be customized to an arbitrary web application in a short period of time. Students will be given one testbed web application and a list of exploits to test their fuzzers. To evaluate their fuzzer, students will be given one two-hour block to test their fuzzer on a real web application that was developed in a web application development course.

**Outcomes**

The overall objective of this course is to give future software engineers an awareness of security risks and mitigations. Students will learn how to design, code, and test for security in their software as that software is being built. Specifically, the course will give students the ability to:

- Apply contemporary formal mathematical modeling techniques to model and analyze the security of a software system.
- Identify project security risks; select and follow risk management strategies.
- Use statistical methods to collect and analyze metrics for assessing and improving the security of a product, process, and project objectives.
- Describe and discuss security concerns at multiple levels of abstraction.
- Comply with data privacy and security requirements when designing a software system.

- Design a software solution for secure access and protection of data.

- Use quality assurance activities and strategies that support early vulnerability detection and contribute to improving the development process.

## 6. Project Results

There are three distinct inputs to the evaluation of this project: (a) external industrial and academic evaluators who work in the cryptography area, (b) the PIs working on the project, and (c) the students who take the courses. These three constituencies are the most appropriate ones to evaluate our work. In the remainder of this section, we comment on our evaluation results to date for two of the courses in this project.

### Cryptography Course Surveys

To propagate the information about previously discussed curriculum changes and our two new courses, we prepared an attitudinal survey and administrated it several times to students enrolled in the Cryptography I course. In this survey, we asked a number of questions regarding the students' prior experience with cryptography, reasons for enrolling in this course and their future academic and career interests in this area.

We are pleased that most of the students expressed a high interest and support for our efforts. There was an overwhelming agreement that the upgraded Cryptography I increased students' interest in cryptography after taking the course and would possibly assist with employment in the security area. The following are some exemplary quotes from students:

> *I am excited to see the possibility of a multiple discipline 'cluster', since a single term only enables one to view a specific element of the field of cryptography in any comprehensive, meaningful way.*
>
> *I think that developing the multi-disciplinary course cluster will be extremely beneficial to the students that choose to participate in it. In my opinion, combining applied cryptography, secure design and implementation, hardware-software co-design, and performance aspects would set our students ahead of most schools in this discipline and make the program much more competitive. I am 100% for developing this course cluster.*
>
> *I believe that it is a great idea since most students on the software side rarely have a chance to interact with hardware at such a low level. It would be a great opportunity to gain better understanding of computer design and experience with systems that have much lower amounts of resources available.*

As expected, most students in Cryptography I are from the Computer Science program, with several from both the Computer Engineering and Software Engineering programs.

**Hardware and Software Design for Cryptographic Applications Course Results**

Prior to the start of the Hardware and Software Design for Cryptographic Applications course, we had one of our external project reviewers examine the material prepared for this course. His positive review is summarized by the following comment:

> *The curriculum is well structured to allow students to build on their individual disciplines with broader skills by designing and implementing real cryptographic sub-systems. The program is logically constructed to first familiarize the student with development platforms and then learn how to use the platforms together with state-of-the-art development tools.*

At the end of the first offering of this course we developed and administrated a new and more elaborate survey to a body of 12 students with 10 responses. The intent here was to collect data that would help us assess achievement of the course learning objectives and measurable outcomes in our first of the two new courses. The main results are summarized in Tables 3 and 4.

Table 3 – Student feedback on course learning objectives

| Learning objective | Strongly agree | Agree | Undecided | Disagree | Strongly disagree |
|---|---|---|---|---|---|
| Provide knowledge and understanding for design and implementation of cryptographic primitives on FPGA-based embedded systems | 40% | 60% | 0% | 0% | 0% |
| To provide knowledge and understanding of hardware-software co-design methodologies and techniques | 60% | 40% | 0% | 0% | 0% |

Table 4 – Student feedback on course learning outcomes

| Learning outcomes | Strongly agree | Agree | Undecided | Disagree | Strongly disagree |
|---|---|---|---|---|---|
| Students have successfully customized and implemented an FPGA based embedded processor system; students understand how to configure linker scripts and build software projects for cryptographic primitives | 70% | 30% | 0% | 0% | 0% |
| Students know how to profile software applications to identify performance bottlenecks. Students have successfully optimized a software application to improve performance; students have analyzed cost in terms of the application size | 50% | 50% | 0% | 0% | 0% |
| Students have successfully performed high level synthesis from C programs to FPGA hardware; students have analyzed performance improvement in a hardware-software system | 70% | 30% | 0% | 0% | 0% |

| | | | | | |
|---|---|---|---|---|---|
| Students have successfully implemented a custom hardware accelerator for selected architecture to achieve desired performance cost/area | 70% | 30% | 0% | 0% | 0% |

In this survey, we also asked additional questions to gain insight into student's general interests in applied cryptography, their self-assessment on background preparation and achievements in this course. The main results referring to this part of the survey are presented in Table 5.

Table 5 – Additional student feedback

| Additional questions | Strongly agree | Agree | Undecided | Disagree | Strongly disagree |
|---|---|---|---|---|---|
| The amount I learned was worth the time invested in this course | 70% | 30% | 0% | 0% | 0% |
| My preparation was adequate for taking this course | 60% | 30% | 10% | 0% | 0% |
| I would recommend to a friend to take this course as an elective | 70% | 30% | 0% | 0% | 0% |
| This course increased my interest in computer security | 10% | 80% | 0% | 0% | 10% |
| I plan to seek employment in the computer security area | 20% | 0% | 50% | 10% | 20% |

## 7. Conclusions

Security concerns are multi-dimensional and require that computing professionals understand topics from hardware, software, and computing theory. The required knowledge and skills span the range from cryptography, to hardware and mixed hardware-software design and performance measurement, to software design and vulnerabilities. Typically, courses that introduce students to these topics are separated into individual disciplines with no effort to show the interrelationships. We have designed a cluster of applied, security-oriented courses that brings the disciplines and knowledge together using cryptography as the motivating application area. We have introduced students to these cross-disciplinary areas by successfully adding performance considerations to an existing cryptography course, developing and teaching a hardware and software design course, and designing a course for software security. The initial results show that students are engaged with the new material, and our results to date have achieved course goals. While cryptography is clearly security-oriented, it is not the only application area that could use our techniques. Computer graphics or computer gaming, to name two other areas, could be the application area that motivates the investigation of combined hardware-software performance and the design of secure software. The same broad multi-disciplinary approach that this project uses is needed to achieve performance and security design requirements in those areas.

## 8. References


[1] National Institute of Standards and Technology (NIST), "Specification for the Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication 197, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] R. Kayser, "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family," Federal Register, vol. 72, no. 212, November 2, 2007. http://csrc.nist.gov/groups/ST/hash/documents/FR Notice Nov07.pdf

[3] National Institute of Standards and Technology (NIST), Computer Security Division, Cryptographic Hash Project Website. http://csrc.nist.gov/groups/ST/hash/index.html

[4] D. Stinson, *Cryptography: Theory and Practice*, CRC Press, third edition, 2006.

[5] P. Schaumont, "A Senior Level Course in Hardware/Software Codesign," IEEE *Transactions on Education, Special Issue on Micro-Electronic Systems Education,* 51(3):306-311, August 2008.

[6] Y. Dandass, "Teaching Application Implementation on FPGAs to Computer Science and Software Engineering Students," *Computers in Education Journal*, Vol. 18, No. 1, January 2008.

[7] E. B. Fernandez, S. Huang, and M. M Larrondo-Petrie. "A set of courses for teaching secure software development," in 19th Conference on Software Engineering Education and Training Workshops, 2006. CSEETW '06, 23- 23. IEEE, 2006.

[8] C. D. Mano, L. DuHadway, and A. Striegel. "A Case for Instilling Security as a Core Programming Skill," in Frontiers in Education Conference, 36th Annual, 13-18. IEEE, 2006.

[9] J. Schumacher, and D. Welch. "Educating leaders in information assurance," *IEEE Transactions on Education* 45, no. 2 (May 2002): 194-201.

[10] B. Endicott-Popovsky, and D. A. Frincke. "A case study in rapid introduction of an information assurance track into a software engineering curriculum," in 17th Conference on Software Engineering Education and Training, 2004. Proceedings, 118- 123. IEEE, 2004.

[11] A. J. A. Wang, "Security testing in software engineering courses," in Frontiers in Education, 2004. FIE 2004. 34th Annual, F1C- 13-18 Vol. 2. IEEE, 2004.

[12] MITRE Corporation, Top 25 Common Weakness Errors. http://cwe.mitre.org/top25.

[13] G. McGraw, *Software Security: Building Security In*, Addison-Wesley Professional, 2006.

[14] C. Dougherty, K. Sayre, R. C. Seacord, D. Svoboda, and K. Togashi, "Secure Design Patterns," *Technical Report, CMU/SEI-2009-TR-010 Cert Program*, Oct. 2009.

[15] http://www.microsoft.com/security/sdl

[16] http://alloy.mit.edu

[17] http://www.parasoft.com/jsp/products/jtest.jsp

[18] http://www.wireshark.org

[19] http://nvd.nist.gov/cvss.cfm