Programming Projects for Undergraduate Information Security Education

Mohamed Aboutabl
aboutams@jmu.edu

James Madison University
Harrisonburg, Virginia, USA

Abstract – Incorporating security mechanisms at the foundation of contemporary software systems has become mandatory for many applications. Universities must empower graduating software engineers with the necessary system / network security education and programming skills that various software developing houses expect. In this paper, I discuss the design and implementation of a set of pedagogical programming projects that supplement an undergraduate semester-long introductory course on information security, which I helped design at my institution. These projects introduce the students to the use of security software libraries in order to implement a diverse array of security mechanisms such as encryption, key exchange, and message authentication / integrity checking. These projects gradually increase in complexity as the semester progresses, and provide an opportunity for follow-up capstone projects suitable for honor classes and/or independent studies.

Keywords

Security education, openSSL, symmetric-key cryptography, RSA public-keys, key exchange, key distribution center, digital signature, user authentication, big integer arithmetic, Diffie-Hellman, MITM attack.

1. INTRODUCTION

The need for solid undergraduate curricula in cyber security has been growing over the last decade. At <my institution name>, we developed a 9-credit curriculum covering an array of diverse topics in computer and network security. The introductory course, titled "Information Security", covers topics such as symmetric- and public-key cryptography, secure message digests and digital signatures, key exchange and user authentication protocols, securing the perimeter of computer networks, and transport-layer / network-layer security protocols (i.e. TLS and IPsec). As a prerequisite to this class, students must complete courses in Data Structures and Computer Organization.

In an effort to infuse simple, yet practical, programming experience into this introductory course, I developed a sequence of programming min-projects that directly map to its learning objectives. In the following sections, I present the learning objectives and the design of these programming projects.

2. PREREQUISITES, LEARNING OBJECTIVES, AND DESIGN APPROACH

The students should have a solid programming experience, specifically in the C language. Prior exposure to the concepts of dynamic memory allocation / deallocation is very valuable in completing the projects. In some of these projects, the students will create multi-process programs and utilize some mechanism for inter-process communication. When needed, the instructor is encouraged at that point to introduce Linux process forking and

Linux pipes. Before designing the programming projects, I conducted a survey of the available cryptographic libraries for possible use in my class[1]. I eventually choose OpenSSL[2] due to its widespread availability, and its sufficient, yet not perfect, online documentation and tutorials. Next, I identified the following top-level learning objectives for my students to achieve after completing the intended programming activities. Specifically, the students will be able to use a cryptographic library to:

i. *apply symmetric-key cryptography to protect the confidentiality of arbitrary data,*
ii. *use public-key cryptography for the exchange of session keys,*
iii. *validate the integrity of messages and authenticate their senders,*
iv. *launch a simple attack to exploit a vulnerable protocol, and*
v. *master the use of big integer arithmetic to implement various cryptographic protocols.*

I designed the programming activities to be at increasing levels of sophistication as follows:

i. *Scaffolded lab activities in which examples of fully functional code are provided to the students as illustrations of the use of the cryptographic library. The students are individually required to study then make small modifications to the provided source code. Completing these activities as an individual, each student gains the confidence and programming skills necessary for a meaningful contribution to the team projects that will follow. I designed two such lab activities as described later in sections 3 and 6.*
ii. *Moderate projects in which teams of 2-to-3 students create code from scratch to develop a single cryptographic mechanism. I designed three such projects described later in sections 5, 7, and 8. Working in teams provides the opportunity for mutual code review among the members of each team.*
iii. *Large project(s)[1] in which the students integrate the skills they have acquired so far in order to build a multi-mechanism security service. At this stage, each student should complete these projects alone so that the instructor is able to make a final assessment of the student's proficiency.*

As these programming projects progress, students will incrementally develop their personal cryptographic "library" and end up with a portfolio of useful routines they can use in their future career.

In the following sections, 3 through 9, I discuss the specific programming activities. A set of project description documents as well as suggested model implementations of these projects are available for academic use by directly contacting the author. I am currently designing some semester-long capstone advanced projects for those students who successfully complete the introductory Information Security course. The students will be expected to implement some sophisticated security service as an Honor course option, or even as a 3-credit independent study. Ideas for such capstone projects are introduced in section 10.

3. LAB ONE: INTRODUCTION TO SYMMETRIC-KEY CRYPTOGRAPHY

This lab illustrates the basic concepts encountered in the symmetric-key encryption of a small memory-resident message. By the end of this lab activity, students will learn:

- *how to generate random symmetric session keys, which are then exchanged via some secure "physical" method,*
- *how to start and clean up the Envelope (**EVP**) high level interface of the **Libcrypto** library of*

[1] *The number of such large projects depends on time availability during the semester.*

OpenSSL,

- *the process of symmetric encryption / decryption using **EVP** interface, and*
- *the use of simple Binary Input/Output (**BIO**) functions to dump binary data in hexadecimal format.*

The operation of two helper functions is explained to students in class, as prototyped in Listing 1. The students learn how to use a cipher context to specify the encryption algorithm and the key material. Using illustrations as the one shown in Figure 1, they also gain an understanding of the process of using the encrypt-update and encrypt-final functions and the possible need for padding. A shell script is also provided to demonstrate the compilation and execution of encrypting and the decrypting agents. At this point, the agents are implemented as two programs running in sequential order. The ciphertext is exchanged via an intermediate binary file. The secure

```
unsigned encrypt( uint8_t *pPlainText, unsigned plainText_len, uint8_t *key,
                  uint8_t *iv, uint8_t *pCipherText ) ;
// Encrypts the plaint text stored at 'pPlainText' into the caller-allocated memory at 'pCipherText'.
// Caller must allocate sufficient memory for the cipher text. The caller-provided 'key' and 'iv' are
// used in some symmetric algorithm, e.g. AES in CBC mode. Returns size of the cipher text in bytes

unsigned decrypt( uint8_t *pCipherText, unsigned cipherText_len, uint8_t *key,
                  uint8_t *iv, uint8_t *pDecryptedText) ;
// Reverses the operation of the encrypt function above
```

Listing 1: Symmetric Encryption / Decryption of Memory-Resident Data

exchange of the encryption key is simulated by physically copying the key from one agent to the other.

In this lab activity, a small plaintext segment, e.g. a string of 50 characters, is encrypted via just one call to encrypt-update. The students are encouraged to analyze the hexadecimal dump of the ciphertext and observe that regardless of the length of the plaintext data, the ciphertext is always a multiple of the block size enforced by the chosen encryption algorithm. Once students complete this lab activity, they are immediately instructed to complete the first programming project described in section 5.

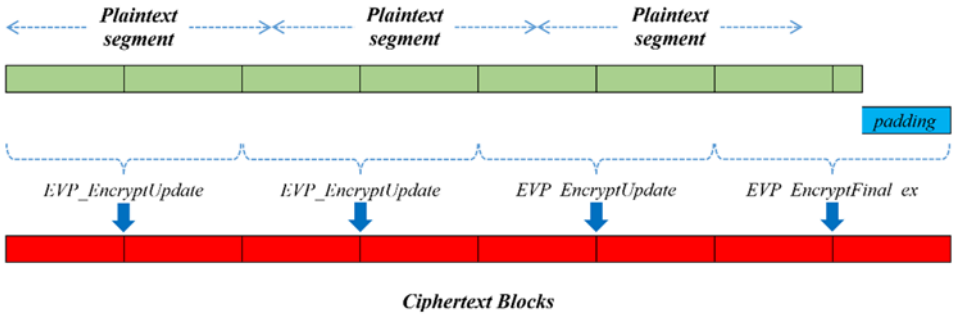

Figure 1: Symmetric-key Encryption using OpenSSL

4. TESTBED FOR THE PROGRAMMING PROJECTS

In most projects, two communicating parties; Amal and Basim[2], use some security mechanism to secure their communication. As shown in Figure 2, the two parties are implemented as concurrently-running processes forked

[2] *Arabic for "Hope" and "Smiley", respectively.*

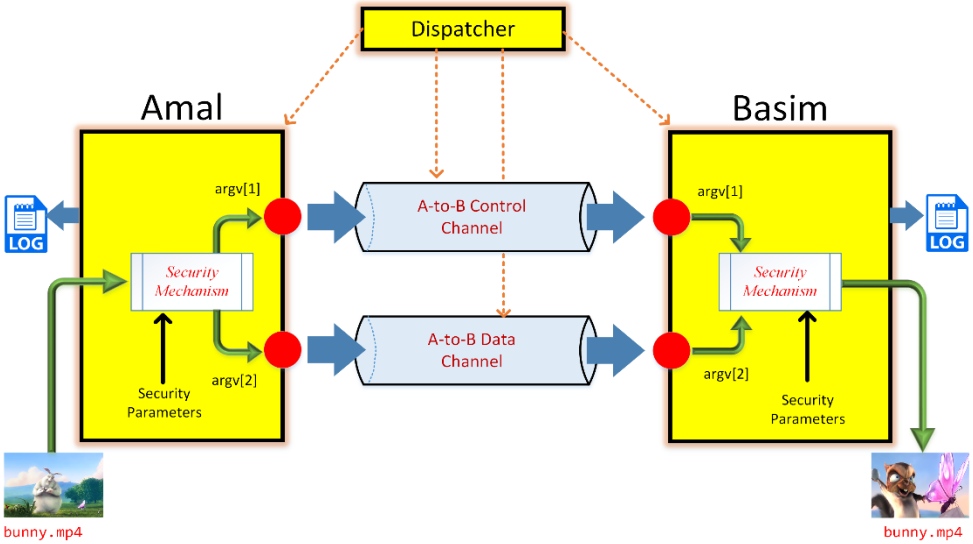

Figure 2: A testbed for the implementation of various security mechanisms

by some parent process; the Dispatcher. The Dispatcher processes also sets up as many communication channels between the two parties as needed for the specific application. These channels can be implemented either as TCP sockets, or as unidirectional Linux pipes. I used pipes in my implementation due to the lack of socket programming experience among the audience of my class. The dispatcher passes the descriptors of the pipes as command-line arguments to the child processes at startup. I used separate channels for the exchange of data and control messages. Once started, the child processes use task-specific security parameters to perform the required security mechanism on the data[3] they exchange. For debugging purposes, each process logs its own actions for later inspection by the students. In order to facilitate grading, and help students validate their code before submission of each project, I provided them with pre-compiled executables of the reference code for the Amal and Basim processes. It is required that the students test the interoperability of their code with the reference code.

5. PROJECT ONE: SYMMETRIC ENCRYPTION OF LARGE DATA

This is the first of a sequence moderately difficult team-programming projects. The students are instructed to read the OpenSSL documentation and discover how to repeatedly invoke the ***`EVP_EncryptUpdate()`*** *function in order to encrypt successive segments of a large plaintext file as shown in Figure* ***1****. Upon production, the ciphertext blocks are immediately sent in real-time to a concurrently running receiver. In contrast, the receiver should repeatedly invokes the* ***`EVP_DecryptUpdate()`*** *function to incrementally decrypt the incoming blocks of cipher text in real-time. Another equally valuable learning objective is to experience the real-life iterative, and sometimes confusing, process of reading through incomplete documentations of a given software package. With very little assistance from the instructor, teams of 2-to-3 students are required to develop two helper functions from scratch, namely* ***`encryptFile()`*** *and* ***`decryptFile()`****, as prototyped in Listing* ***2****. As an exception, the instructor may provide help illustrating the creation of the multi-process environment, and the use of the inter-process communication mechanism of choice. It is reasonable to assume that 6 to 9 student hours are sufficient to complete the task.*

[3] *I used the freely-downloadable Big Buck Bunny mpeg video from https://peach.blender.org/*

```
int encryptFile( int fd_in, int fd_out, unsigned char *key, unsigned char *iv ) ;
// Encrypt the incoming data stream from 'fd_in' file descriptor using the caller-provided symmetric 'key'
// and initial vector 'iv'.  The resulting ciphertext is forwarded to the 'fd_out'  file descriptor.
// Returns actual size in bytes of the ciphertext

int decryptFile( int fd_in, int fd_out, unsigned char *key, unsigned char *iv ) ;
// Decrypt the incoming ciphertext stream from 'fd_in' file descriptor using the caller-provided symmetric
// 'key' and initial vector 'iv'.  The resulting plaintext is forwarded to the 'fd_out'  file descriptor.
// Returns actual size in bytes of the plaintext
```

Listing 2: Helper Functions for Project One

6. LAB TWO: PUBLIC-KEY CRYPTOGRAPHY AND KEY EXCHANGE

This individually-completed lab illustrates the use of public-key cryptography to exchange session keys. By the end of this lab activity, each student will learn how to:

- *generate and handle RSA key pairs, then extract the public key component using the openSSL command-line tool.*
- *extract RSA key pairs from disk files into a C program,*
- *encrypt a small block of memory-resident data using RSA public keys, and decrypt using the corresponding private key,*
- *apply RSA public-key cryptography in order to exchange symmetric session keys, and*
- *Apply symmetric-key cryptography to preserve the confidentiality of a large data file.*

For this purpose, the students are introduced to the following library functions: **`RSA_public_encrypt()`** , **`RSA_private_decrypt()`**, **`PEM_read_RSA_PUBKEY()`**, *and* **`PEM_read_RSAPrivateKey()`**. *As in Lab One, the instructor provides the students with the source code and the shell script to execute the lab activity, but in a copy-restricted PDF format. This persuades the students to carefully read and analyze the instructor's code as they manually type it. The file encryption / decryption helper functions developed in Project One are to be re-used in this lab.*

7. PROJECT TWO: MESSAGE DIGESTS & RSA-BASED DIGITAL SIGNATURE

After completing Lab Two, teams of students will next implement a message authentication mechanism to validate the integrity and verify the author of exchanged communication messages [3]. By the end of this project, students will learn how to:

- *calculate the digest of a large file using some secure hashing function,*
- *sign the file digest using the sender's private RSA key, and*
- *verify the integrity of the transferred file and the identity of its sender by validating the signature.*

The students are required to implement a new helper function **`fileDigest`**, *as prototyped in Listing* ***3****, and add it to their growing personal cryptographic "library". This function is used by the sender to calculate the digest of the data stream, while also transmitting a copy of the data itself via the A-to-B Data channel to the recipient. The recipient uses the same function to calculate the digest of the incoming data stream for the purpose of signature validation. In contrast to Lab Two, the sender uses its RSA private key to encrypt the computed digest and send the resulting digital signature via the A-to-B Control channel to the recipient. For this purpose, the*

```
size_t fileDigest( int fd_in , int fd_out , uint8_t *digest) ;
// Read all the incoming data stream from 'fd_in' file descriptor and compute the SHA256 hash value of
// this incoming data into the array 'digest'.  If the file descriptor 'fd_out is > 0, store a copy of the
// incoming data to 'fd_out'. Returns actual size in bytes of the computed hash value
```

Listing 3: Helper Function for Project Two

openSSL functions ***RSA_private_encrypt()*** *and* ***RSA_public_decrypt()*** *are introduced to the students. Using the library function* ***RSA_size()****, the students discovers how to determine the size of the memory needed to store the signature as a function of the size of the RSA keys. This project is worth 6 to 8 student hours.*

8. PROJECT THREE: BIG-INTEGER ARITHMETIC & ELGAMAL DIGITAL SIGNATURE

In this project, the students will use big integer arithmetic to implement a cryptographic protocol. The project is expected to achieve the following learning objectives:

- *To properly use primitives from the BIGNUM suite of big integer operations afforded by openSSL,*
- *To fully understand the details of the Elgamal Digital Signature protocol [3], and*
- *To efficiently use online cryptographic calculators [4][5][6] as a debugging tool for the students' programs.*

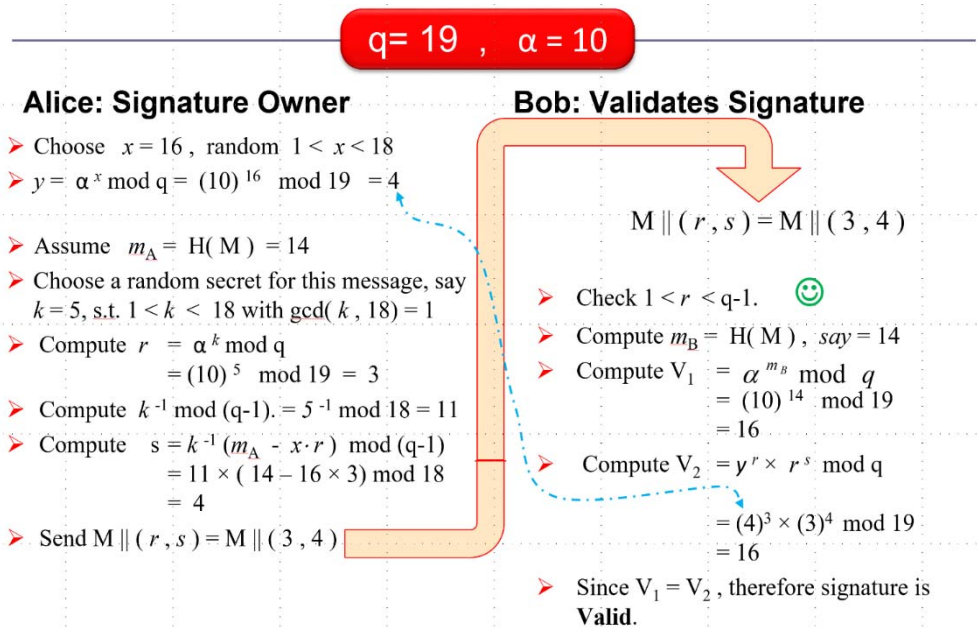

Figure 3: Simple Illustration of Elgamal Digital Signature Scheme

Prior to starting this project, the students are provided with a detailed example, as shown in Figure ***3****, illustrating the calculations involved in generating and validating an Elgamal digital signature. The students are instructed to use online calculators in order to try other examples. Doing so strengthens their understanding of the scheme and the involved concepts of large prime number generation, choice of primitive roots, modular arithmetic, etc. As part of the implementation, big integers of varying sizes are to be sent over the A-to-B Control*

```
int BN_write_fd( const BIGNUM *bn  , int fd_out) ;
// Sends the #of bytes , followed by the bytes themselves of a BIGNUM to file descriptor fd_out
// Returns 1 on success, 0 on failure

BIGNUM * BN_read_fd( int fd_in ) ;
// Read the #of bytes , then the bytes themselves of a BIGNUM from file descriptor fd_in
// Returns: a newly-created BIGNUM, which should be freed later by the caller, or NULL on failure

BIGNUM * BN_myRandom( const BIGNUM *p ) ;
// Returns a newly-created BIGNUM such that: 1 < BN < (p-1)

void elgamalSign( const uint8_t *digest , int len , const BIGNUM *q , const BIGNUM
        *gen , const BIGNUM *x , BIGNUM *r , BIGNUM *s , BN_CTX *ctx) ;
// Use the prime 'q', the primitive root 'gen' and the private 'x'
// to compute Elgamal signature (r,s) into the 'len'-byte long 'digest'

int elgamalValidate( const uint8_t *digest , int len ,  const BIGNUM *q , const BIGNUM
        *gen , const BIGNUM *y , BIGNUM *r , BIGNUM *s , BN_CTX *ctx ) ;
// Use the prime 'q', the primitive root 'gen',  and the public 'y'
// to validate the Elgamal signature (r,s) on the 'len'-byte long 'digest'
```

Listing 4: Helper Functions for Project Three.

channel to share some security parameters with the recipient. Special helper functions, prototyped in Listing ***4****, are required to send and receive such data properly. The primary functions to generate and validate the signature are also shown in Listing* ***4****. This project requires 10 to 12 student hours to complete.*

9. INTEGRATION PROJECTS

Near the end of the semester, each individual student is required to develop a programming project that integrates many of the skills learned through the earlier team projects. Two alternative projects are discussed in the following subsections. Both are based on course material the students learn through the introductory Information Security course [3]. The students are expected to spend 15 to 20 hours completing the project of their choice.

9.1. Two-Way User Authentication & Key Exchange Protocol

In this choice of the final project, the students extensively use symmetric key encryption and some big integer arithmetic. Two communicating parties ***A****mal and* ***B****asim seek the help of a trusted third party, the* ***K****ey* ***D****istribution* ***C****enter, to provide them with a session key and to authenticate their identities to each other. Prior to exchanging any confidential data with each other,* ***A****mal and* ***B****asim activate a version of Needham-Schroeder protocol slightly-enhanced to add two-way identity authentication [3]. The students first analyze this protocol in class to comprehend its operation and discuss its vulnerability to playback attacks.*

The three parties; Amal, Basim, and the KDC are to be implemented as concurrent processes using four control pipes to carry the five protocol message shown in Figure ***4****. A data channel is also needed to carry the encrypted movie file from Amal to Basim. The students are encouraged to use a Length-Value structure to represent each*

protocol object, for example the simple object ID_A or the nested compound object $E_{K_b}(K_s || ID_A)$. This is due to their variable-size nature.

Message	From-To	Content
1)	A ⟶ KDC:	$ID_A \,||\, ID_B \,||\, N_a$
2)	KDC ⟶ A:	$E_{K_a}\left(K_s \,||\, ID_B \,||\, N_a \,||\, E_{K_b}(K_s \,||\, ID_A)\right)$
3)	A ⟶ B:	$E_{K_b}(K_s \,||\, ID_A) \,||\, N_{a2}$
4)	B ⟶ A:	$E_{K_s}(f(N_{a2}) \,||\, N_b)$
5)	A ⟶ B:	$E_{K_s}(f(N_b))$

Figure 4: Enhanced Needham-Shroeder User Authentication and Key Exchange

9.2. Man-in-the-Middle Attack on Diffie-Hellman Key Exchange

In this choice of the final project, the students extensively big integer arithmetic to implement the Diffie-Hellman public-key cryptographic protocol, as well as some symmetric-key cryptography for data exchange. The two communicating parties Amal and Basim use the anonymous version of the DH protocol [3] for a session key exchange, which they intended to later use for encrypting/decrypting a data message M[3]. However, their protocol exchange is intercepted and modified by Darth, a malicious hidden attacker, as shown in Figure ***5****. Again, all three principals are to be implemented as concurrent processes that use Linux pipes for protocol control and data messages. The successful demonstration of a student's project requires his/her Darth process to intercept the DH exchange of two instructor-provided pre-compiled executables for Amal and Basim and successfully*

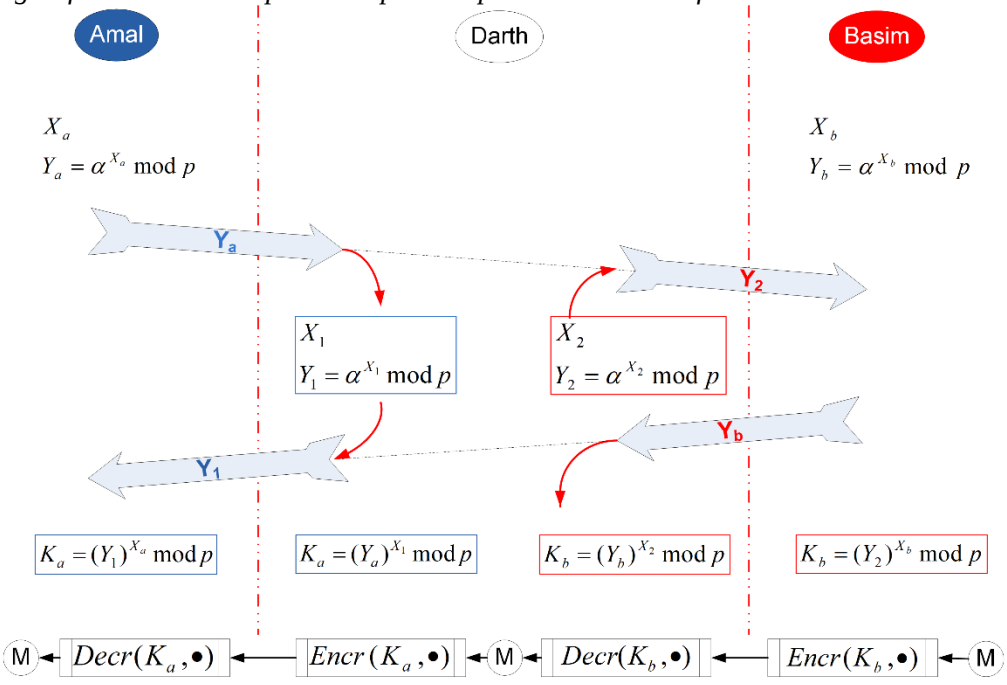

Figure 5: Man-In-The-Middle Attack on Diffie-Hellman Key Exchange

compromise the confidentiality of the encrypted data message M. As a recommended reading, the students are referred to [7] for possible remedies to such MITM attack on the DH exchange.

10. CONCLUSION AND FUTURE CAPSTONE PROJECTS

By the end of the semester, several students requested the opportunity to develop projects that are more advanced. I am currently creating the requirement documents for few such projects to be adopted in Spring 2019 as an independent study offered to the top 10% of those who completed this course. The student will be required to build systems for applications such as:

- *A simplified Pretty Good Privacy protocol for messages and data files, and*
- *Simulation of a simple distributed electronic voting system.*

It is worth noting that the programming labs and projects described in this paper were adopted in two sections of the introductory Information Security course at my institution with about 60 students enrolled. As expected, students encountered some difficulty during the first programming project. However, by the semester, more than 83% of the students successfully completed the final integration project. At the time, most students were close to graduation, and, therefore, were actively hunting for jobs in the industry. Several of them indicated that during their job interviews they received very favorable comments from the prospective employers after the students presented the portfolio of the projects they completed in this course. This confirmed that the skill set acquired from these projects is very well aligned with the contemporary needs of the security industry. I plan to further refine these projects and expand their scope to incorporate more security mechanisms.

REFERENCES

[1] Comparison of Cryptographic Libraries. https://en.wikipedia.org/wiki/Comparison_of_cryptography_libraries Last accessed 3/19/2018.

[2] OpenSSL home page. https://wiki.openssl.org/index.php/Main_Page Last accessed 3/19/2018.

[3] William Stallings. Cryptography and Network Security: Principles and Practice (7th Edition). Pearson (2017).

[4] Paul Trow. Big Integer Modular Arithmetic Calculator. http://ptrow.com/perl/calculator_bigint.pl (2017)

[5] The Blue Tulip. Primitive Root Calculator. http://www.bluetulip.org/2014/programs/primitive.html (2009)

[6] FileFormat.Info Hash Functions Calculator. http://www.fileformat.info/tool/hash.htm

[7] A. S. Khader and D. Lai. Preventing man-in-the-middle attack in Diffie-Hellman key exchange protocol. 22nd International Conference on Telecommunications (ICT), Sydney, NSW, 2015, pp. 204-208.

[8] Nance K., Taylor B., Dodge R., Hay B. (2013) Creating Shareable Security Modules. In: Dodge R.C., Futcher L. (eds) Information Assurance and Security Education and Training. WISE 2009. IFIP Advances in Information and Communication Technology, vol 406. Springer, Berlin, Heidelberg

[9] Schweitzer D., Gibson D., Bibighaus D., Boleng J. (2013) Preparing Our Undergraduates to Enter a Cyber World. In: Dodge R.C., Futcher L. (eds) Information Assurance and Security Education and Training. WISE 2009. IFIP Advances in Information and Communication Technology, vol 406. Springer, Berlin, Heidelberg

[10] Van Niekerk J., Goss R. (2013) Towards Information Security Education 3.0. In: Dodge R.C., Futcher L. (eds) Information Assurance and Security Education and Training. WISE 2009. IFIP Advances in Information and Communication Technology, vol 406. Springer, Berlin, Heidelberg

[11] Bishop M. (2013) Some "Secure Programming" Exercises for an Introductory Programming Class. In: Dodge R.C., Futcher L. (eds) Information Assurance and Security Education and Training. WISE 2009. IFIP Advances in Information and Communication Technology, vol 406. Springer, Berlin, Heidelberg

[12] Van Niekerk J., von Solms R. (2013) Using Bloom's Taxonomy for Information Security Education. In: Dodge R.C., Futcher L. (eds) Information Assurance and Security Education and Training. WISE 2009. IFIP Advances in Information and Communication Technology, vol 406. Springer, Berlin, Heidelberg

[13] John Viega; Matt Messier; Pravir Chandra. Network Security with OpenSSL. O'Reilly Media, Inc. (2002)