Proceedings of the 14th Colloquium for Information Systems Security Education
Baltimore Marriott Inner Harbor
Baltimore, Maryland June 7 - 9, 2010

Putting It All Together: Theory and Practice in Courses on Cryptography

Philip Scheffler, Michael Hylkema, Anatoly Temkin, *Boston University*

Abstract – The Computer Science Department at Boston University Metropolitan College offers a sequence of two graduate courses on cryptography. Being mathematical in nature, they lay down a solid foundation of knowledge that can be utilized in semester projects. Two projects are designed which tie together the concepts from both courses to implement real world scenarios in public key infrastructure and web of trust modeling. Several cryptographically secure algorithms are required to be implemented by students to successfully complete these projects. Upon finishing these two courses along with the corresponding projects detailed in this paper students will not only grasp the mathematical foundations of cryptography but also be ready for implementing the learned concepts and techniques in the industry environment.

I. Introduction

In this paper we present two comprehensive cryptography projects developed by information security students in the Computer Science department at Boston University Metropolitan College. The projects are based on the cumulative knowledge developed throughout two semester long courses on cryptography.

The projects were introduced during the second course following the study of symmetric and asymmetric ciphers, cryptographic hash functions, message authentication, and relevant cryptanalysis methods. Introduction to these topics was not limited to only implementation details. General concepts from number theory were introduced as well. Number theory topics covered included the theory of finite groups, rings and fields, with special emphasis placed on the group of integers modulo a prime and the field of polynomials modulo an irreducible polynomial. A strong understanding of the number theory concepts was required to complete the projects. It was instrumental in implementing the public and private key cipher algorithms required for the projects.

Computer Science Department, Metropolitan College, Boston University 808 Commonwealth Ave, Room 250 Boston, MA 02215

Additional topics also covered prior to project introduction were prime number testing and pseudo-random number generation algorithms. The knowledge of these topics was necessary for the development of public key cipher algorithms.

In parallel with the projects' implementation, the second semester course also covered applied information security topics such as authentication protocols, key management protocols, network security, and public key infrastructures. Basic knowledge of these topics helped to solidify an understanding of the projects and their purpose in application.

An interested reader can find a detailed description of topics covered in the two courses on cryptography in *Teaching Cryptography to Continuing Education Students* [1], *Integrated Curricula for Computer and Network Security Education* [2], and *Laboratory Assignments in Security Education* [3].

Coursework did not only cover theory but also application through small programming projects directly related to each theoretical topic. Students gained experience programming algorithms for pseudo-random number generators, prime number testing, factorization of composite numbers, finding discrete logs, the RSA and Diffie-Hellman key exchange protocols following the introduction of each topic. The acquired programming experience was invaluable for the successful completion of the projects in the second course.

Not all students in the class had a strong programming background. The programming projects helped inexperienced students to develop stronger programming skills. These skills were fully utilized in their work on the projects for the second course.

Each student selected a programming language to complete their projects. Language selection was ad-hoc with students selecting languages that they either knew very well or wanted to learn. This resulted in multiple

programming languages used by the class. When projects were completed students would compare their programming experience with other students. Some languages turned out to be superior to others as far as their efficiency for coding algorithms was concerned. These languages were adopted by most students for the second semester projects.

Each individual project helped develop understanding for a particular section of the course and also led students to want to see how individual projects fit together. Students taking the second course could not wait to apply the previously learned programming skills and knowledge of the first course to the second course.

Finally, requiring students to implement algorithms as separate building blocks for the final projects turned out to be a good pedagogical approach as it prevented students from feeling overwhelmed. For students without a strong programming experience, individual smaller projects are easier to implement than one large comprehensive project with many interconnected pieces. The implementation of individual algorithms during the course prepares students for the smooth transition to the implementation of the comprehensive final project. At the final project implementation, students focus on the best ways to put the individual projects together rather than develop the whole project from scratch.

II. General Project Information

Initial preparation for the comprehensive final project begins midway through the second course to allow enough time for project completion. Based on the knowledge gained from the first course, students should have enough theoretical and practical background to feel comfortable to start putting the individual algorithm building blocks together in their implementation of the final project.

Students are divided into groups for the final project and they meet to discuss the requirements of the project. There is no mandate for a group size. In our project, group size varied from three people in one group to five people in the other. The only requirements for a successful completion of the project are the knowledge of information security fundamentals and successful utilization by each student of cryptographic algorithms learned in the two courses.

Project implementation time is approximately 4 weeks. A review of algorithms required for the project and group selection happens in week 1. The project implementation and testing occurs during weeks 2 and 3. The full two weeks are recommended if implementing algorithms based on a field of binary strings. There are currently very few libraries supporting fields of binary strings and therefore students' groups were responsible for coding the implementations. The project execution phase occurs in week 4

The first project is a weak Public Key Infrastructure (PKI) implementation. The project is designed to demonstrate the effects of an adversary recovering the private key of a subject in a centralized trust model. The term weak represents the implemented 32 bit RSA modulus rather than a more typical 1024 bit or larger modulus. The 32 bit size allows for factorization in a reasonable time.

The second project is a strong Pretty Good Privacy (PGP) like implementation. The project is designed to demonstrate the development of a decentralized trust model. The term strong represents the implemented 1024 bit RSA modulus to prohibit private key recovery. A value of 1024 bits was selected based on contemporary literature and should be adjusted to match current industry specifications.

Both projects require implementation of two common cryptographic algorithms: a public-private key algorithm, such as RSA or Diffie-Hellman and a cryptographically secure hash function such as Whirlpool or MD5. Each project also requires a third cryptographic algorithm not common amongst projects. In addition, the weak PKI project requires an appropriate public-private key cryptanalysis algorithm, such as Pollard's Rho or Baby-Step Giant Step, and the PGP project requires a symmetric key cipher such as AES or Whirlpool. It is important to note that there is a hash function as well as a symmetric key cipher based on the Whirlpool algorithm.

Algorithm implementations are not limited to a particular programing language. Projects can be successfully completed in Ruby, Python, C, C++ and Java. It is our experience that Python and Ruby allow for the shortest implementation time by both experienced and inexperienced programmers. Both languages have native big integer support, dynamic typing, and an interactive interpreted mode.

III. Project 1: A weak Public Key Infrastructure for Message Authentication and Message Forging

The first project is a PKI. It demonstrates the message authentication process and PKI vulnerabilities. This project requires one student to act as the Certificate Authority (CA) for all other students. CA implementation can be very simple, with the selected student manually tracking all other student certificate data, or more complex with the student implementing a web based CA automating certificate management.

Each student is required to select a public/private key algorithm for key generation, a hash algorithm for message hashing, and an algorithm to perform cryptanalysis on other student's public/private key algorithm. It is not necessary that all students select the same algorithms. However, each student must be able to verify all received messages and break the public/private key algorithm for all other students.

For the purpose of demonstration, we chose to implement a PKI utilizing a web based CA, the RSA algorithm for key generation, the Whirlpool hash function for message hashing, and Pollard's Rho to break the RSA modulus. The public/private key algorithm, hashing function, and algorithm for cryptanalysis were common across all students.

The CA web architecture in this demonstration used Ruby on Rails with a MySQL database to store certificate data. All cryptographic algorithms for the CA were implemented in Ruby. Hardware support for the certificate authority was minimal and adequately fulfilled by a desktop uniprocessor computer running Linux.

Other student implementations of the algorithms used many different programming languages. The selection of languages were Java, C++, C, Ruby, and Python. However, a majority of the group used Python because its ease of use for algorithm implementation.

A. Key and Certificate Generation

Students should generate their own public/private key pair. Key size should be large enough to require computational assistance to recover the private key, but also small enough to be recoverable in a reasonable time. Reasonable time is a relative phrase and should be defined based on course deadlines.

In our demonstration implementation we selected an RSA modulus of 32-bits. We also tried a modulus of 64 and 100 bits but discovered that these moduli took longer to create and factor. A modulus above 100 bits should be avoided as factoring a modulus of this size in a reasonable amount of time requires implementing an advanced factorization algorithm not covered in the course.

Next, students should request a certificate from the CA. The request contains their name, email address and public key information. In response the CA generates a public key certificate signed with its private key. The CA should augment student supplied data with a unique ID and time stamp of the signature request.

The CA calculates its certificate signature by hashing the certificate data and encrypting the hash with its private key. A valid certificate contains the unique ID, student's name, email address, public key algorithm information, and signed signature hash. The certificate should then be sent to the requesting student and also stored by the certificate authority. This process is shown in Diagram 1.1.

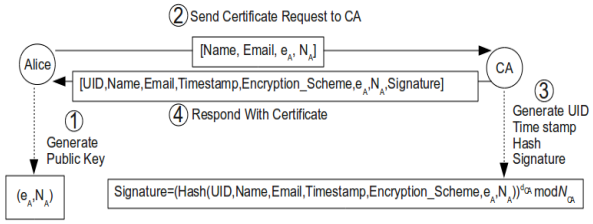

Diagram 1.1 : Certificate Request Process

Our implementation followed this certificate format and process. The certificate format was derived from the X.509 standard and could be more closely aligned with it depending on future project designs.

B. Message Exchange

After initialization, each student prepares a message to send to another student of their choice. The message is hashed using the student's hash function and signed using their private key. As an optional step the sender may include a copy of their public key certificate to the receiver for message verification. If the sender does not include a public key certificate, the receiver can obtain the certificate from the certificate authority. The sending process can be seen in Diagram 1.2 along with the verification method for the message.

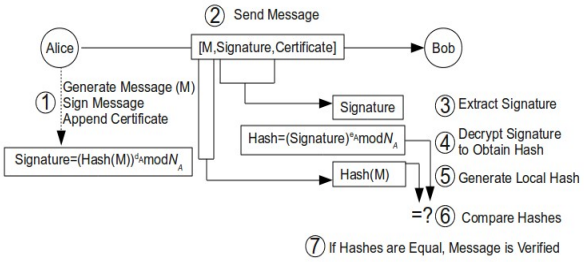

Diagram 1.2 : Sending and Verifying Message

The weak key size requirement causes a problem during message hashing and signing. Modern cryptographic hash functions create digests larger than the maximum recommended project key size. Depending on the selected public/private key algorithm it might be necessary to modify the generated hash value.

For example a requirement for the RSA algorithm is all messages be less than and relatively prime to the modulus. The Whirlpool hash function produces digests of 512 bits. Given the weak key size requirement of the PKI project, the Whirlpool hash function will violate the RSA modulus requirement. A solution to this problem is to select only the first 31 bits of the hash value before signing.

This trivial solution adds an additional vulnerability for research by students. The smaller digest space makes it easier to find message collisions. Our implementation did not attempt to analyze this flaw because of time constraints.

Upon receipt of a message the receiver will also perform verification of the CA. This process is depicted in Diagram 1.3.

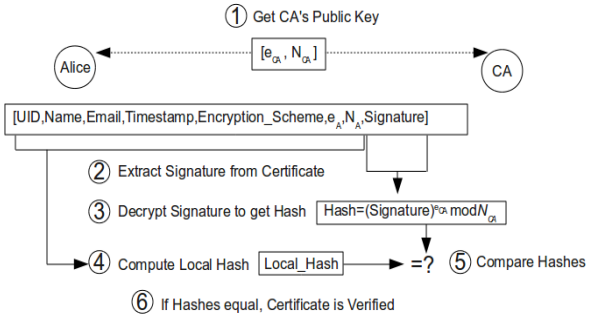

Diagram 1.3 : Certificate Verification

It may prove easiest for students to use email as the message channel between them. If this method is chosen, only the email message body should be in scope for hashing, all other information, such as headers, should be ignored. This approach does create a problem however. During the forgery round a recipient could easily recognize if a received message was a forgery or legitimate based on the individual the email came from. While a minor issue, it may remove the enjoyment of not knowing which messages were forged.

As a suggestion to solve this problem, the CA or another student could be used as an anonymous re-mailer. It would receive email messages from students, capture the message body, and send the message to the original recipient. Forging attacks would be far more interesting because a recipient would not be tipped off by the email header information if the message received was legitimate or forged.

C. The Fun Part

The project now requires each student to perform two forgeries. First forging the signature of another student and second forging the signature of the certificate authority. This is where our requirement that the implemented public/private key algorithm can be broken in a reasonable amount of time comes into play.

The first forgery requires that the forger obtains the certificate for another student. As all the certificates were available from the CA this is trivial. The public key information of the other student is then extracted from their certificate and attacked with the appropriate algorithm. Once this is done the forger can create messages which will authenticate as though they were from the other student. Diagram 1.4 illustrates the first forgery.

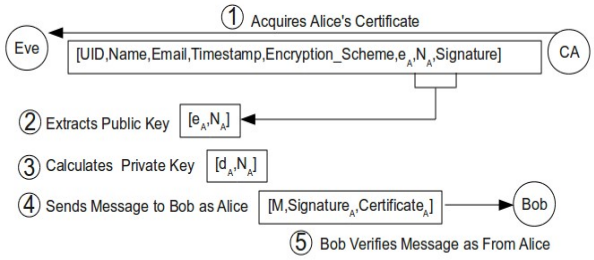

Diagram 1.4 : Message Forgery

The second forgery is similar to the first. The CA's public certificate is obtained by simply requesting it from the CA. Public key information is extracted and the private key recovered. The attacker can now create forged signed certificates of all students.

This completes the first lab on public key infrastructure. At this point students should submit source code for all implemented algorithms as well as some other proof of project success such as hard or soft copies of legitimate and forged messages. A demonstration by groups of students may also be appropriate.

IV. Project 2: A Strong PGP-like Infrastructure for Message Authentication

The second project is a Pretty Good Privacy (PGP) like implementation showing web of trust creation and message authentication. This project is a decentralized trust model, requiring each student to maintain their own public keyring. Students must participate in a key signing party with other students to build their own web of trust.

A common public/private key algorithm and cryptographically secure hash algorithm should be used for all students to simplify project implementation. In addition, a symmetric key algorithm is required to encrypt each student's private key file. The symmetric key algorithm however does not have to be common between students.

In our demonstration implementation we chose to implement the RSA algorithm for public/private key development, the Whirlpool hash function for message hashing, and the Whirlpool cipher using Cipher Block Chaining mode for private key file encryption. We found that it was easiest to do all of our implementations using Python and combined them into a common command line tool.

Unlike the first project, no central public key database is required for this project. This is what differentiates this project from the first one and makes it more PGP in nature, which is more ad-hoc, rather than PKI, which is more structured. To demonstrate this, students should be encouraged to form links with each other as needed and find paths to other students where there exists no common link.

A. Key and Ring Generation

Each student should generate a public/private key pair using the group's selected algorithm. In our demonstration implementation we chose the RSA algorithm with a 1024 bit modulus. A 1024 bit modulus was selected because the project was designed to be as “real world” functional as possible.

Keyring development requires each student to create two files, a private key file and a public key ring file. The private key file should be encrypted using the student's selected symmetric key algorithm. The public key file does not require any encryption but should be hashed and signed by the student's private key. These steps can be seen illustrated in diagram 2.1

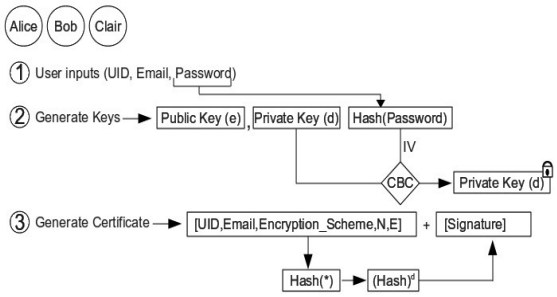

Diagram 2.1 : Generating Key Files and Self Signed Certificates

The private key file stores the student's private key used for document signing. It has no special format. The public key ring file stores the student's own public key and other students' public keys in the student's web of trust. Each entry in the public key ring should contain a student identifier, a public/private key algorithm identifier, the public key algorithm data, self signed certificate hash and message hash algorithm identifier.

An example public key ring format may use a student user ID, student email address, public/private key RSA algorithm identifier, student RSA modulus, student RSA public key, self signed certificate hash, Whirlpool hash algorithm identifier.

To generate the public key ring, students should import another student's self signed public key information. A review of the public key file after each import should show who a student trusts. It is not necessary to have each student import all other students' public key certificates. It is better to have two disjoint groups so that the web of trust fails for some message exchanges. Importing of certificates is shown in Diagram 2.2 below.

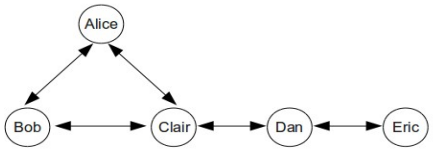

Diagram 2.2 : Signed Certificate Exchange

B. Message Exchange

Students should exchange messages in four ways. First each student should create a message and validate it against their own public key certificate. This tests to ensure that all algorithms are implemented correctly. Second each student should send a message to another student, for whom he has their public key certificate. This demonstrates a direct relationship in the web of trust. Third each student should generate a message to another student not in their public key ring but one or two hops away. This demonstrates an indirect relationship in the web of trust. Finally, each student should generate a message to another student who is outside of the web of trust. This demonstrates that validation cannot occur. Diagram 2.3 shows both message creation and validation.

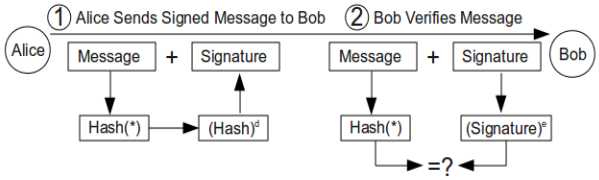

Diagram 2.3 : Message Creation and Validation

It is recommended that students exchange plain text files. All normal conversations between students should be signed and verified by the web of trust. This demonstrates to students the ease of introducing authentication between parties.

C. Demonstrating Security

To demonstrate the effectiveness of the project, students should generate a message, sign the message, and edit the signed message but not resign the edited message. Each student should select a receiver at random and send the forged message to the receiver. The message verification might fail if the sender and receiver are not in the same web of trust, but will definitely fail during comparison of hash values.

During this part of the project, students should record each step performed during message signing and verification. We think that this helps to reinforce the principles of the project and provides a way for students to review their results with each other.

Another method to demonstrate security is to attempt to factor the RSA modulus. Although computationally infeasible, implementation of the RSA modulus might be incorrect allowing recovery of the private key. Of course the upper bound for testing is relatively high, so it is impractical to consider this be a definitive demonstration of security. However this approach does stimulate an interest in reviewing known factorization algorithms.

V. CONCLUSION

The goal of the two projects is to provide information security students with an outlet for the knowledge gained in the two cryptography courses offered by the Computer Science department at Boston University Metropolitan College. The individual and cumulative projects combine abstract course fundamentals with practical experience to enable students to develop applicable knowledge required in today's information security field. The projects reinforced classroom material and allowed students to actively participate in the learning process.

VI. REFERENCES

1. A.Temkin: Teaching Cryptography to Continuing Education Students *Proceedings of the Fifth World Conference on Information Security Education*
June 19-21, 2007 West Point, NY

2. Zlateva, T et al. Integrated Curricula for Computer and Network Security Education, *Proceedings of the Colloquium for Information Systems Security Education, Society for Advancing Information Assurance and Infrastructure Protection*, Washington, D.C., June 2003.

3. Chitkushev, L.T. et al. Laboratory Assignments in Security Education, *Proceedings of the 4th World Conference on Information Security Education*. Editors: Natalia Miloslavskaya, Helen L. Armstrong. Success Through Information Security Knowledge, IFIP TC11 / WG11.8 Forth World Conference on Information Security Education (WISE 4), June 18-20, 2005, Moscow, Russia. ISBN 5-7262-0565-0