Security Programming with High-Level Abstractions: a Tutorial (Extended Abstract)*

Paolo Modesti
Department of Computing, Engineering and Technology
University of Sunderland, UK

Abstract

The specification of security protocols with high-level programming abstractions, suited for security analysis and verification, has been advocated by the formal methods for security research community. Based on these principles of application design, we developed a tutorial to introduce undergraduate students to the foundations of security programming. The main pedagogical goal of this tutorial is to teach, in a simple and effective way, how to build secure distributed applications using common cryptographic primitives abstracting from their low-level details. The tutorial is aimed at helping the students to grasp quickly the main security concepts and to apply them effectively to the coding of distributed programs implementing security properties like authentication and secrecy. As programming is one of the main skills required by the cybersecurity industry, we believe that this tutorial can contribute to the professional development of future graduates.

1 Introduction

The implementation of security protocols is an important building block for the construction of safe and robust applications. In particular, security protocols play a key role in protecting user data exchanged over a network infrastructure that can be assumed to be under adversary control, as in the Dolev-Yao attacker model [1]. Programming security protocols is challenging and error-prone, as it requires that the high-level security properties of a protocol must often hard-coded explicitly, in terms of low-level cryptographic notions.

As experience has shown, low-level implementation bugs are discovered even in protocols like TLS and SSH which are widely used and thoroughly tested. Therefore, the specification of security protocols with high-level programming abstractions, suited for security analysis and automated verification, has been

*Presented at the HEA National Conference on Learning and Teaching in Cybersecurity, Birmingham, UK, 15th June 2016

advocated by the formal methods for security research community [2, 3]. This was one of the main reasons for developing tools for verification of security protocols in the symbolic model [4, 5] and for the automatic generation of security protocols implementations [6, 7, 8]. In order to widen the adoption of these tools among practitioners, some of them allow for the specification of security protocols with simple languages based on the Alice and Bob notation [9, 10, 11, 12]. This makes the coding considerably simpler and more compact than their equivalent in formal languages like the process calculi.

Based on these principles of application design and on the availability of existing tools [7, 4], at the Department of Computing, Engineering and Technology of the University of Sunderland, we developed a tutorial to introduce undergraduate students to the foundations of security programming. This tutorial was part of the CET324 Advanced Cybersecurity module (Year 3, academic year 2015-16).

The main pedagogical goal of this tutorial is to teach, in a simple and effective way, how to build secure distributed applications using common cryptographic primitives (symmetric and asymmetric encryption, digital signature, hashing, message authentication codes) abstracting from their low-level details. The tutorial is aimed at helping students to quickly grasp the main security concepts and to effectively apply them to the coding of distributed programs implementing security properties like authentication and secrecy. This constructivist approach allows students to put into practice what each student is learning, perceiving his/her progress building increasingly more complex software artefacts. As programming is one of the main skills required by the cybersecurity industry, we believe that this tutorial can contribute to the professional development of our future graduates.

2 Pedagogical Approach

This tutorial was developed and delivered as part of the CET324 module where students needed to develop their understanding and ability to implement secure computer systems by identifying and evaluating the security considerations required for each stage of the computer system development life-cycle. A particular focus is given to the role of cryptography in secure systems design. Therefore, students were introduced to the basic principles of cryptography and how these could be applied to secure systems design (including encryption, decryption, ciphers, digital signatures and public key infrastructures).

In more detail, the tutorial addresses the following:

- learning need: as our students comes from different programs (computer science, computer forensics, network computing, etc.), their programming abilities are mixed. Therefore, the traditional learning curve to master cryptography is likely to be too steep for many students;
- purpose: in a limited amount of time (3 hours of lectures, 4 hours of supervised tutorial, and 10-12 hours of independent work) students should

be able to show a good understanding of secure systems design principles, in particular in the implementation of simple security protocols, making use of cryptographic primitives like symmetric and asymmetric encryption, digital signatures and message digest;

- activities: a series of learning activities with increasing levels of difficulty in bite size steps, and incremental elaboration, help the student to manage the increasing complexity of the material being delivered. Activities are detailed in "Tutorial Structure and Content" section;

- outcomes: critical understanding of the principles and applications of cybersecurity and secure systems design, identifying and minimising the security risks, effectively implementing in reliable and effective way simple security protocols;

- evaluation: as this tutorial was run in the laboratory, we used a formative evaluation strategy with one to one feedback from the tutor. Ongoing testing/evaluation was also intrinsic to the code presented in the activities, helping to motivate students to progress through the materials, incrementally improving their coding ability as the module progressed.

3 Tutorial Structure and Content

The tutorial is based on the Java programming language and on the cryptographic services offered by the Java Cryptography Architecture (JCA). We chose them because they are widely used by the industry and are freely available. In particular, we used the AnBxJ library which is part of the AnBx Compiler and Code Generator [7], a tool developed as an academic research project. Instead of directly using the JCA programming interface, the AnBxJ library wraps, in an abstract way, the JCA interface and implements the custom classes necessary to encode programs in Java. The AnBxJ library offers a high degree of generality and customization, since the API does not commit to any specific cryptographic solution (algorithms, libraries, providers). Moreover, the library provides access in an abstract way to the communication primitives used to exchange messages in the standard TCP/IP network environment. For the modelling and verification, we used the OFMC model checker [4], a tool for symbolic security protocol analysis that supports the Alice and Bob notation.

The pre-requisites for this tutorial include the knowledge of the basic principles of Object-Oriented Programming. Most of our students learned C# in their previous years of studies. No prior knowledge of security programming was assumed. The basic knowledge of cryptographic primitives is acquired during the module lectures that are part of this tutorial.

The main topics covered are (unless otherwise stated the tools used are Java and the AnBxJ library):

- Client/Server programming

- Key generation (tool: JDK `keytool`)
- Simple secret exchange, based on asymmetric cryptography
- Weak authentication using digital signature
- Strong authentication using challenge/response and digital signature
- Implementation of a secure channel using secrecy and authentication
- Defensive programming, checks on receptions
- Configuration of cryptographic algorithms
- Modelling and Verification of Security Protocols specified in the Alice and Bob notation (tool: OFMC)

The tutorial is structured in a series of tasks of increasing difficulty. Each programming task usually starts with a piece of code that students have to analyse and run. As the AnBxJ library allows running the application with different debugging levels, log messages are available to report about every steps of the programs execution. Students are then asked to modify or extend the program in order to add some functionality or to achieve some security goals, thus applying and reinforcing their learning.

4 Evaluation and Conclusion

We ran this tutorial for the first time in the second semester of the academic year 2015-16 and used ongoing student feedback to refine and improve the module. So far, the major achievement was that students were able to build simple applications implementing security goals like authentication and secrecy. Another important aspect is that students approached for the first time an advanced topic like modelling and verification of security protocols. This offered them the opportunity to become aware of the existence of methodologies and techniques that are not only used in academic research but also are becoming increasingly adopted by the industry. This may be useful to help student in pursuing their postgraduate studies or to have better chances to find a qualified job in the software development and cybersecurity sector.

Acknowledgements The author wishes to thank Susan Jones and Alastair Irons for their constructive feedback.

References

[1] D. Dolev and A. Yao, "On the security of public-key protocols," *IEEE Transactions on information Theory* **2**(29), 1983.

[2] M. Bugliesi and R. Focardi, "Language based secure communication," in *Computer Security Foundations Symposium, 2008. CSF'08. IEEE 21st*, pp. 3–16, 2008.

[3] M. Avalle, A. Pironti, and R. Sisto, "Formal verification of security protocol implementations: a survey," *Formal Aspects of Computing* **26**(1), pp. 99–123, 2014.

[4] D. Basin, S. Mödersheim, and L. Viganò, "OFMC: A symbolic model checker for security protocols," *International Journal of Information Security* **4**(3), pp. 181–208, 2005.

[5] B. Blanchet, B. Smyth, and V. Cheval, "ProVerif 1.94: Automatic cryptographic protocol verifier, user manual and tutorial," 2016.

[6] M. Avalle, A. Pironti, D. Pozza, and R. Sisto, "JavaSPI: A framework for security protocol implementation," *International Journal of Secure Software Engineering* **2**(4), pp. 34–48, 2011.

[7] P. Modesti, "AnBx: Automatic generation and verification of security protocols implementations," in *8th International Symposium on Foundations & Practice of Security, LNCS* **9482**, Springer, 2015.

[8] O. Almousa, S. Mödersheim, and L. Viganò, "Alice and Bob: Reconciling formal models and implementation," in *Programming Languages with Applications to Biology and Security: Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*, C. Bodei, G.-L. Ferrari, and C. Priami, eds., *Lecture Notes in Computer Science* **9465**, pp. 66–85, Springer International Publishing, 2015.

[9] S. Mödersheim, "Algebraic properties in Alice and Bob notation," in *International Conference on Availability, Reliability and Security (ARES 2009)*, pp. 433–440, 2009.

[10] M. Bugliesi and P. Modesti, "AnBx-Security protocols design and verification," in *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security: Joint Workshop, ARSPA-WITS 2010*, pp. 164–184, Springer-Verlag, 2010.

[11] D. Basin, M. Keller, S. Radomirovic, and R. Sasse, "Alice and Bob meet equational theories," in *Logic, Rewriting, and Concurrency*, N. Martí-Oliet, P. C. Ölveczky, and C. Talcott, eds., *Lecture Notes in Computer Science* **9200**, pp. 160–180, Springer International Publishing, 2015.

[12] M. Bugliesi, S. Calzavara, S. Mödersheim, and P. Modesti, "Security protocol specification and verification with AnBx," *Journal of Information Security and Applications* **30**, pp. 46–63, 2016.