Type-Driven Development of Security Protocols

Jan de Muijnck-Hughes  
University of Glasgow  
Jan.deMuijnck-Hughes@glasgow.ac.uk

Edwin Brady  
University of St Andrews  
ecb10@st-andrews.ac.uk

Abstract
Implementing security protocols correctly is hard; verifying them is too. However, we also implement them in languages different to those we verify them in. With dependent types we can reason on software with greater precision and link specifications to implementations. In this talk we examine how state-of-the-art programming language research can help us to verify and implement protocols in the same language.

Keywords  Dependent Types, Session Types, Verification, Security Protocols, Algebraic Effects, Usability

1. The Importance of Usable Cryptography
Cryptography is a fundamental technology that provides strong mathematical guarantees towards the security of data and is used in practice to provide, for example: secure storage and transmission of sensitive data; copyright protection of intellectual property; and authentication to computer networks. When combined with telecommunication protocols and the increased distribution of data, a class of system emerges that looks to provide these strong mathematical guarantees towards the security of data that is in transit as well as rest.

However, the usability problems associated with cryptography extended to the design of the software APIs of libraries that provide cryptographic functionality. ‘Why Johnny can’t Encrypt’ is a seminal paper that describes the usability issues for software programs facilitating the use of cryptography [12]. A badly designed and documented API will increase the likelihood that said API could be misused. For example, when we examine the man pages for OpenSSL one of the de facto crypto-libraries, the plethora of options, functions, and data structures overload the user with information and choices. Unfortunately, well-known security incidents have further shown that flaws in the implementation and deployment of cryptographic mechanisms can have a far reaching and disastrous effect upon society at large. Given the importance of these mechanisms as used in software systems the correctness of their implementation and use is paramount when seeking to protect our data. To that end tools such as ProVerif/CryptoVerif, EasyCrypt, and Cryptol have been developed [2, 4, 7] that facilitate formal reasoning [1, 5] on cryptographic protocols and their implementations. Further, NaCL is an attempt by leading cryptographers to provide a usable API for common cryptographic operations [3] that restricts the choices and options for developers. Other, similar, approaches look at developing semantic APIs that offer similar restrictions [9].

However, the composition and use of functions within these APIs are not restricted. Developers can still misuse the presented API. With these tools there is also a separation of concerns between the specification, verification, and implementation of communication protocols. The tooling used to present the specification is different from the tooling used to implement and verify the protocols.

2. Dependently Typed Sessions
In dependently typed languages such as Idris [6] types are treated as first class language constructs, depend on values, and be computed, facilitating precise reasoning on programs. Such languages provides programmers with a highly expressive environment in which the precise properties of a program can be specified and reasoned upon directly within the type system.

Session Types are a typing discipline to describe interactions between communicating processes agnostic to the medium of communication. Work by Honda et al. [8] has shown how session types provide a means to specify, formally, the interactions of a communicating system and provide guarantees over the session’s description and its realisation in software.

Our current research is directed towards how state-of-the-art programming language research into Dependent Types and Session Types [6, 8] can be directed towards verified implementations of security protocols and mechanisms. Specifically we are interested in introducing new frameworks for machine checkable verification of protocol specification implementations using these state-of-the-art programming language techniques. With new frameworks, protocol designers can verify protocol specifications using the same tooling for protocol implementation. No longer is the tool of verification separate from the tool of specification from the tool of implementation. By removing the gap between the tooling we hope that better, more verified, protocol implementations can be constructed.

1 https://www.openssl.org/docs/manpages.html
3. The Sessions EDSL

Our project looks first to investigate how to reason about communication protocols and link them to their implementations, bridging the gaps between protocol verification, specification, and implementation. Sessions is a specification language for communication protocols that allows developers to detail interactions between multiple entities agnostic to the context of the communication. We have designed Sessions as an Embedded Domain Specific Language (EDSL) in Idris, the same language we will implement the protocols in.

Figure 1 demonstrates how Sessions can describe and reason about protocol specifications by modelling an insecure variant of the Symmetric Key Kerberos authentication protocol taken from Tanenbaum and Steen [11, Chapter 9]. This protocol describes how two trusted third parties (a Timestamp Service and Key Authority) are used to establish a secure connection between two users. Key to the protocol’s operation is the construction and communication of secret keys and tickets between the various parties in various different messages.

However, we do not as yet have a means to inspect and reason about cryptographically secured messages to the same depth as seen in ProVerif [4], nor a means to link specifications with implementations. We are actively working towards these goals, and how they can be addressed within Sessions. We envisage that algebraic effects can be used to provide compile time guarantees that an implementation of a protocol satisfies it’s specification [10]. Further, we expect that programming language research can help further known techniques [5] to reason about the contents of cryptographic messages.

References
