The Design and Verification of a Cryptographic Security Architecture

Peter Gutmann

A thesis submitted in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy
in
Computer Science

The University of Auckland, 2000

Abstract

A cryptographic security architecture constitutes the collection of hardware and software which protects and controls the use of encryption keys and similar cryptovariables. This thesis presents a design for a portable, flexible high-security architecture based on a traditional computer security model. Behind the API it consists of a kernel implementing a reference monitor which controls access to security-relevant objects and attributes based on a configurable security policy. Layered over the kernel are various objects which abstract core functionality such as encryption and digital signature capabilities, certificate management, and secure sessions and data enveloping (email encryption).

The kernel itself uses a novel design which bases its security policy on a collection of filter rules enforcing a cryptographic module-specific security policy. Since the enforcement mechanism (the kernel) is completely independent of the policy database (the filter rules), it is possible to change the behaviour of the architecture by updating the policy database without having to make any changes to the kernel itself. This clear separation of policy and mechanism contrasts with current cryptographic security architecture approaches which, if they enforce controls at all, hardcode them into the implementation, making it difficult to either change the controls to meet application-specific requirements or to assess and verify them.

To provide assurance of the correctness of the implementation, this thesis presents a design and implementation process which has been selected to allow the implementation to be verified in a manner which can reassure an outsider that it does indeed function as required. In addition to producing verification evidence which is understandable to the average user, the verification process for an implementation needs to be fully automated and capable of being taken down to the level of running code, an approach which is currently impossible with traditional methods. The approach presented here makes it possible to perform verification at this level, something which had previously been classed as “beyond A1” (that is, not achievable using any known technology).

The versatility of the architecture presented here has been proven through its use in implementations ranging from 16-bit microcontrollers through to supercomputers, as well as a number of unusual areas such as security modules in ATMs and cryptographic coprocessors for general-purpose computers.

Acknowledgements

This thesis has been a long time in coming. My thesis supervisor, Dr.Peter Fenwick, had both the patience to await its arrival and the courage to let me do my own thing, with occasional course corrections as some areas of research proved to be more fruitful than others. I hope that the finished work rewards his confidence in me.

I spent the last two years of my thesis as a visiting scientist at the IBM T.J.Watson Research Centre in Hawthorne, New York. During that time the members of the global security analysis lab (GSAL) and the smart card group provided a great deal of advice and feedback on my work, augmented by the considerable resources of the Watson research library. Leendert van Doorn, Paul Karger, Elaine and Charles Palmer, Ron Perez, Dave Safford, Doug Schales, Sean Smith, Wietse Venema, and Steve Weingart all helped contribute to the final product, and in return probably found out more about lobotomised flatworms and sheep than they ever cared to know.

Before coming to IBM, Orion Systems in Auckland, New Zealand for many years provided me with a place to drink Mountain Dew, print out research papers, and test various implementations of the work described in this thesis. Paying me wages while I did this was a nice touch, and helped keep body and soul together.

Portions of this work have appeared both as refereed conference papers and in online publications. Trent Jaeger, John Kelsey, Bodo Möller, Brian Oblivion, Colin Plumb, Geoff Thorpe, Jon Tidswell, Robert Rothenburg Walking-Owl, Chris Zimman, and various anonymous conference referees have offered comments and suggestions which have improved the quality of the result. As the finished work neared completion, Charles “lint” Palmer, Trent “gcc –wall” Jaeger and Paul “lclint” Karger went through various chapters and pointed out sections where things could be clarified and improved.

Finally, I would like to thank my family for their continued support while I worked on my thesis. Without them, none of this would have been possible.

Hawthorne, July 2000 Peter Gutmann

John Roebling had sense enough to know what he *didn't* know. So he designed the stiffness of the truss on the Brooklyn Bridge roadway to be *six times* what a normal calculation based on known static and dynamic loads would have called for. When Roebling was asked whether his proposed bridge wouldn't collapse like so many others, he said "No, because I designed it six times as strong as it needs to be, to prevent that from happening"

— *Jon Bentley, "Programming Pearls"*

Preface

A cryptographic security architecture constitutes the collection of hardware and software which protects and controls the use of encryption keys and similar cryptovariables. Traditional security architectures have concentrated mostly on defining an application programming interface (API) and left the internal details up to individual implementers. This thesis presents a design for a portable, flexible high-security architecture based on a traditional computer security model. Behind the API it consists of a kernel implementing a reference monitor which controls access to security-relevant objects and attributes based on a configurable security policy. Layered over the kernel are various objects which abstract core functionality such as encryption and digital signature capabilities, certificate management, and secure sessions and data enveloping (email encryption). This allows them to be easily moved into cryptographic devices such as smart cards and crypto accelerators for extra performance or security. Chapter 1 introduces the software architecture and provides a general overview of features such as the object model and inter-object communications.

Since security-related functions which handle sensitive data pervade the architecture, security must be considered in every aspect of the design. Chapter 2 provides a comprehensive overview of the security features of the architecture, beginning with an analysis of requirements and an introduction to various types of security models and security kernel design, with a particular emphasis on separation kernels of the type used in the architecture. The kernel contains various security and protection mechanisms which it enforces for all objects within the architecture, as covered in the latter part of the chapter.

The kernel itself uses a novel design which bases its security policy on a collection of filter rules enforcing a cryptographic module-specific security policy. The implementation details of the kernel and its filter rules are presented in Chapter 3, which first examines similar approaches used in other systems and then presents the kernel design and implementation details of the filter rules.

Since the enforcement mechanism (the kernel) is completely independent of the policy database (the filter rules), it is possible to change the behaviour of the architecture by updating the policy database without having to make any changes to the kernel itself. This clear separation of policy and mechanism contrasts with current cryptographic security architecture approaches which, if they enforce controls at all, hardcode them into the implementation, making it difficult to either change the controls to meet application-specific requirements or to assess and verify them. The approach to enforcing security controls which is presented here is important not simply for aesthetic reasons but also because it is crucial to the verification process discussed in Chapter 5.

Once a security system has been implemented, the traditional (in fact pretty much the only) means of verifying the correctness of the implementation has been to apply various approaches based on formal methods. This has several drawbacks which are examined in some detail in Chapter 4. This chapter covers various problems associated not only with formal methods but with other possible alternatives as well, concluding that neither the application of formal methods nor the use of alternatives such as the CMM present a very practical means of building high-assurance security software.

Rather than taking a fixed methodology and trying to force-fit the design to fit the methodology, this thesis instead presents a design and implementation process which has been selected to allow the design to be verified in a manner which can reassure an outsider that it does indeed function as required, something which is practically impossible with a formally verified design. Chapter 5 presents a new approach to building a trustworthy system which combines cognitive psychology concepts and established software engineering principles. This combination allows evidence to support the assurance argument to be presented to the user in a manner which should be both palatable and comprehensible.

In addition to producing verification evidence which is understandable to the average user, the verification process for an implementation needs to be fully automated and capable of being taken down to the level of running code, an approach which is currently impossible with traditional methods. The approach presented here makes it possible to perform verification at this level, something which had previously been classed as “beyond A1” (that is, not achievable using any known technology). This level of verification can be achieved principally because the kernel design and implementation has been carefully chosen to match the functionality embodied in the verification mechanism. The behaviour of the kernel then exactly matches the functionality provided by the verification mechanism and the verification mechanism provides exactly those checks which are needed to verify the kernel. The result of this co-design process is an implementation for which a binary executable can be pulled from a running system and re-verified against the specification at any point, a feature which would be impossible with formal-methods-based verification.

The primary goal of a cryptographic security architecture is to safeguard cryptovariables such as keys and related security parameters from misuse. Sensitive data of this kind lies at the heart of any cryptographic system and must be generated by a random number generator of guaranteed quality and security. If the cryptovariable generation process is insecure then even the most sophisticated protection mechanisms in the architecture won't do any good. More precisely, the cryptovariable generation process must be subject to the same high level of assurance as the kernel itself if the architecture is to meet its overall design goal, even though it isn't directly a part of the security kernel.

Because of the importance of this process, an entire chapter is devoted to the topic of random number generation for use as cryptovariables. Chapter 6 begins with a requirements analysis and a survey of existing generators, including extensive coverage of pitfalls which must be avoided. It then describes the method used by the architecture to generate cryptovariables, and applies the same verification techniques used in the kernel to the generator. Finally, the performance of the generator on various operating systems is examined.

Although the architecture works well enough in a straightforward software-only implementation, the situation where it really shines is when it is used as the equivalent of an operating system for cryptographic hardware (rather than having to share a computer with all manner of other software, including trojan horses and similar malware). Chapter 7 presents a sample application in which the architecture is used with a general-purpose embedded system, with the security kernel acting as a mediator for access to the cryptographic functionality embedded in the device. This represents the first open-source cryptographic processor, capable of being built from off-the-shelf hardware controlled by the software which implements the architecture.

Because the kernel is now running in a separate physical device it is possible for it to perform additional actions and checks which are not feasible in a general-purpose software implementation. The chapter covers some of the threats which a straightforward software implementation is exposed to, and then examines ways in which a cryptographic coprocessor based on the architecture can counter these threats. For example it can use a trusted I/O path to request confirmation for actions such as document signing and decryption which would otherwise be vulnerable to manipulation by trojan horses running in the same environment as a pure software implementation.

Finally, the conclusion looks at what has been achieved, and examines avenues for future work.

Contents