# Adaptive Security

Thesis by

Christiaan Johan Lamprecht

In Partial Fulfilment of the Requirements
for the Degree of
Doctor of Philosophy


School of Computing Science
Newcastle University
Newcastle upon Tyne, UK

April 2012

# Abstract

Automated runtime security adaptation has great potential in providing timely and fine grained security control. In this thesis we study the practical utility of a runtime security-performance trade off for the pervasive Secure Socket Layer (SSL/TLS) protocol. To that end we address a number of research challenges.

We develop an Adaptive Security methodology to extend non-adaptive legacy security systems with adaptive features. We also create a design of such an extended system to support the methodology. The design aids in identifying additional key components necessary for the creation of an adaptive security system.

We furthermore apply our methodology to the Secure Socket Layer (SSL) protocol to create a design and implementation of a practical Adaptive SSL (ASSL) solution that supports runtime security adaptation in response to cross-cutting environmental concerns. The solution effectively adapts security at runtime, only reducing maximum server load by 15% or more depending on adaptation decision complexity.

Next we address the security-performance trade off research challenge. Following our methodology we conduct an offline study of factors affecting server performance when security is adapted. These insights allow for the creation of policies that can trade off security and performance by taking into account the expected future state of the system under adaptation. In so doing we found that client SSL session duration, requested file size and current security algorithm play roles predicting future system state. Notably, performance deviation is smaller when sessions are longer and files are smaller and vice versa. A complete Adaptive Security solution which successfully demonstrates our methodology is implemented with trade-off policies and ASSL as key components. We show that the solution effectively utilises available processing resources to increase security whilst still respecting performance guarantees.

# Publications

"Runtime Security Adaptation Using Adaptive SSL"

C.J. Lamprecht and A.P.A. van Moorsel, *14th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE Computer Society, Taiwan, pp.305-312, 2008

"Adaptive SSL: Design, Implementation and Overhead Analysis"

C.J. Lamprecht and A.P.A. van Moorsel *In First International Conference on Self-Adaptive and Self-Organizing Systems*, SASO 2007, 9-11 July 2007, Cambridge, Massachusetts, Di Marzo Serugendo, G., Martin-Flatin, J.-P., Jelasity, M. et al. (eds.) pp 289-292 IEEE Computer Society, 2007. Notes : submission for Applications Track.

"Investigating the efficiency of cryptographic algorithms in online transactions"

C.J. Lamprecht, A.P.A van Moorsel, P. Tomlinson, and N. Thomas. *International Journal of Simulation: Systems, Science & Technology* Vol. 7, Issue 2, pp 63-75 United Kingdom Simulation Society, 2006

"Performance Measurement of Web Services Security Software"

C.J. Lamprecht and A.P.A van Moorsel, *In 21st UK Performance Engineering Workshop*, UKPEW 2005, University of Newcastle , 14th/15th July 2005, Thomas, N. (ed.), pp 11-20, University of Newcastle upon Tyne, School of Computing Science, 2005

To my parents,

Heinrich & Carine

# Acknowledgements

Firstly I would like to thank my supervisor Professor Aad van Moorsel for his insight and direction over the course of my PhD. Also many thanks to Graham Morgan and Nigel Thomas for being part of my supervisory committee. Additionally I would also like to thank the examiners, John Fitzgerald and Gordon Blair, for their constructive insights.

My appreciation also to the School of Computing Science for the PhD scholarship as well as the Resilience for Survivability in Information Systems Technology (ReSIST) Network of Excellence for affording me the opportunity to present and discuss my research with partners at other European universities.

Finally I would like to dedicate this thesis to my parents; my father Heinrich and mother Carine, whose continual and self-sacrificial support of my future made this PhD possible. Baie dankie!

# Contents

# List of Figures

# Chapter 1

# Introduction

Since the early days of Computing Science the idea of "separation of concerns" has had a significant impact on how academia and industry approach the design and implementation of software systems, models and architectures. The phrase was coined by Edsger W. Dijkstra in his 1982 paper "On the role of scientific thought":

> We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained -on the contrary!- by tackling these various aspects simultaneously. It is what I sometimes have called *"the separation of concerns"*, which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. [4]

Some notable examples of this idea in practice includes the Open Systems Interconnection (OSI) Basic Reference Model [5]. It provides a set of layered abstractions for computer network protocol design which allows for separation of concerns between the layers and so improves operability between protocols. More recently the influential contributors to the World Wide Web protocol standards have also conceded to this idea and created two languages, namely eXtensible HyperText Markup Language (XHTML) and Cascading Style Sheets (CSS), to separate style from content when designing web pages. Separation of concerns is also made explicit through accepted programming paradigms such as procedural programming and object-oriented pro-

gramming. Aspect Oriented programming pushes this idea even further by allowing cross-cutting concerns, which could include security, logging, tracing, profiling, pooling and cacheing among others, to be dynamically added to objects at runtime (or compile time) without the objects needing to have any knowledge of the particular type of addition. The implications and usefulness of this paradigm is an active topic for current research [6].

That said, the boundaries between concerns is often a grey area. As such it is an interesting area for research, to either push or break these boundaries in aid of increasing understanding and grounding their definition. This is no more so evident than in areas where systems are designed to be intelligent and react to their surroundings. Intelligent systems usually require additional information not generally accepted as being a justified concern. Due to the general nature of this statement it can be applied to a wide variety of different areas of research that may fall under the general area of the thesis topic Adaptive Security. Such research include adaptive access control policies where policies incorporate application specific information in policy decisions [7], adaptive intrusion detection systems which allow individual trust management to conserve processor resources [8], adaptive agents where the system itself moves between different domains and has to detect and adapt to various malicious scenarios [9], adaptive security in resource constrained networks where appropriate security protocols are selected at runtime based on the current network conditions [10, 11] and threats [12], adaptive security infrastructures (ASI) where the ASI consists of many security systems which cooperate to ensure minimal policy conflicts [13, 14] and many more.

Of interest in this thesis are the concerns of system security and system performance, in particular separating these concerns at design time and addressing the contention between them with a intelligent Adaptive Security solution at runtime. The type of security we address is specifically data privacy through cryptography and 'adaptive' implies the ability to change the cryptographic algorithm at runtime through an intelligent trade off policy. This is further discussed below and expounded throughout the thesis.

## 1.1 Terminology

We now take this opportunity to provide definitions of key concepts utilised throughout the thesis.

**Security**

Security is the measure taken as a precaution against dangers or threats. In IT security this is addressed as a composite set of attributes occurring concurrently. Attributes are:

- Confidentiality: limiting disclosure of information to authorised individuals only.
- Integrity: absence of unauthorized system alterations
- Availability: accessible for authorised actions when needed

In this thesis we address all of the above attributes as they apply to the Secure Socket Layer (SSL) protocol in a web server environment. Confidentiality is the protection provided through encryption/decryption of sensitive data. Integrity is provided by SSL through hashing and public key cryptography. Availability is addressed through the trade-off and can protect against Denial of Service attacks by decreasing the security.

**Quality of Service (QoS)**

Quality is measured as the predictability or guarantee of service delivery. In this thesis we measure the percentage of successfully serviced requests, where success implies a service response within a given time period.

**Trade-Off**

Trade-off is defined as *"a balance achieved between two desirable but incompatible features; a compromise."* In this thesis we research the trade-off between Security and QoS as defined above. A Trade-off as defined in this Thesis exhibits a number of characteristics. Firstly it refers to the actions taken to increase or decrease security at the cost or benefit of QoS respectively. Secondly, in the trade-off a minimal level

of both QoS and security is specified. Lastly, the extent and intensity of service usage influences QoS over time and so security and QoS is traded-off to support the maximum achievable level of security at any particular moment in time within the stipulated security and QoS constraints.

### Adaptive Security

Adapt is defined as: *"make (something) suitable for a new use or purpose; modify"* With reference to software systems the type of adaptation is often defined with reference to when it is achieved. Namely, adaptive can mean static adaptation where the system is modified at design, compile or link time. It can also refer to dynamic adaptation where the system modifies its own behaviour at runtime.

With reference to the definition of security above and our understanding of adaptive we consider Adaptive Security as a runtime modification of the cryptographic algorithm employed to secure sensitive data.

## 1.2 Motivation

Choosing an appropriate level of security to protect a system is inherently a trade off exercise in maximising security whilst taking into account financial concerns as well as client Quality of Service (QoS) constraints. We believe that this trade off is often made implicitly in industry and has only recently come to the attention of the research community.

Successfully evaluating security often requires many years of practical experience as such decisions are often based on an array of abstract factors. Factors could include the likelihood of attacks, the strength of attacks, value of the data processed, strength of the security algorithms, trust in the algorithms and many more. Additionally there are even ambiguities in evaluating the actual security algorithms themselves. Current security algorithm evaluation research [15, 16] provides formulae to evaluate the security of algorithms but they depend heavily on subjective input such as level of trust in DES encryption, expected cryptanalytic developments, accuracy of Moore's law and average cost of computing power to name a few.

Furthermore, the chosen level of security also has a direct impact on system performance. Choosing the right level of security to meet QoS guarantees requires a great deal of expertise and foresight. Firstly the expected QoS without security needs to be understood well. This partly depends on the combined performance effect of system configuration parameters, such as buffer sizes and cache expiration policies. Due to the large number of configurable parameters and their complex interdependencies it may be difficult to determine their combined effect on system performance. Secondly, the expected client load on the system is also difficult to predict and plays a role in deciding what QoS guarantees can be provided. Providing security places an additional load on the system which may be difficult to determine at design time. Not only does the additional performance cost depend on the chosen algorithm but also on the chosen algorithm implementation. Even once an algorithm implementation is chosen the performance impact on the system is greatly dependant on how clients use the system, as we will show in this thesis.

Once a security algorithm is chosen there are no guarantees that the external factors on which the decision was based will remain constant. For example, external threat levels are likely to change over time, as will the value of the data that clients store on the system. The number of expected clients as well as their behaviour may also change. New cryptanalytic developments can also severely affect the strength of or trust in a particular algorithm. Making only one security decision can therefore in itself leave the system at risk or break QoS guarantees.

Choosing an appropriate level of security is thus a continual trade off exercise best accomplished at runtime through an Adaptive Security system which can take relevant environmental factors into account before making a security decision.

## 1.3 Goal

In reference to the above motivational factors we formulate our goal as follows:

**GOAL:** **Demonstrate the practical utility of Adaptive Security in trading off security and QoS at runtime.**

In particular we note that due to the subjective and abstract factors that influence a security decision a certain level of threat to the system inevitably remains. In this thesis we focus on reducing this threat further by trading off security and performance to provide better security whilst maintaining QoS constraints. A particular challenge is to use the available resources effectively without overloading the system or breaking client QoS constraints.

Such an automated runtime adaptation has the potential to tap into available resources to maintain a high level of security. It can also provide timely (i.e. in response to threats) and fine grained (i.e. based on data value) security control.

## 1.4 Approach

Solutions to a number of research problems need to be explored to achieve the goal above. Firstly, the issue of how best to create an adaptive security system which can take QoS metrics into account and then adapt security at runtime needs to be addressed. There is also the challenge of trading off security and Quality of Service effectively; addressing issues such as quantifying each in such a way to allow a trade-off, analysing which contextual information most appropriately represents QoS and constructing effective security trade-off policies.

In addressing these questions we focus on creating an adaptive security system by extending legacy security systems with adaptive features. We also choose to consider security systems where part of the trade-off can be studied offline through experimentation and in so doing provide insight as to the expected future QoS state of the system under adaptation.

We therefore first study current literature to elicit the implicit adaptive principles of current adaptive security research and so provide the cornerstone for our methodology. The methodology addresses the challenge of extending legacy security systems with adaptive features and the steps needed to create a trade-off policy. We apply our methodology to the Secure Socket Layer SSL/TLS security protocol as implemented for the industry recognised open source Web Server system called Apache. We conduct an offline study of the security-performance contention in the system resulting

from its SSL security provision to clients and so gain the necessary insight to create a runtime trade-off policy.

In summary, in this thesis we start by investigating the performance overhead of typical security protocol implementations as well as the level of security they provide. We furthermore study the performance effect of utilising such implementations on an Apache based server and study how the performance impact due to security varies under different client-server conditions. Such key insights provide the basis for a security-performance trade-off policy which allows intelligent security adaptation at runtime based on current client-server conditions. To achieve effective runtime security adaptation we follow our methodology to design and implement a novel Adaptive SSL (ASSL) solution.

## 1.5 Contributions

The analysis and tools constituting this thesis provide the following research contributions:

- A methodology detailing key steps and activities to create an adaptive security solution from legacy non-adaptive security systems. Steps include: Establishing a control point to leverage control of the security decision process from the existing system. Identifying a cross-cutting concern which is a measurable property of the environment. Studying the interrelationship between security and the cross-cutting concern in the existing system context by identifying measurable factors which influence their relationship (This is done offline). Formulation of a trade-off goal which is optimised based on multiple objectives, one of which is a security objective.

  The methodology specifically addresses existing security systems. Also, only systems where the security adaptation has a direct impact on the monitored resource are considered as they provide the necessary complexity to study a runtime trade-off. We also note that it must be possible to gain control of the security decision making process from the legacy system for the methodology to be effective.

- Ability to predict the runtime system performance behaviour resulting from a security adaptation. This is achieved through offline experimentation which measures the system performance states under different security algorithms and contextual parameters. Parameters such as the requested file size, session duration and performance cost of the cryptographic algorithms are studied. Offline analysis allows one to systematically address the complexities of such predictions leaving the less resource intensive decisions, i.e. those supported by the offline results, to be made at runtime. Performance outcomes of security decisions can therefore be studied offline before any decisions need to be made at runtime. We are thus able to use performance prediction to divine a trade-off policy that can respect QoS and security guarantees as the outcome of the adaptation is known beforehand.

- Design and implementation of Adaptive SSL for the Apache web server. The Secure Socket Layer (SSL) is augmented with adaptive features through the application of our methodology and adherence to the design which accompanies it. Adaptation concepts such as 'component based design' and 'separation of concerns', as discussed in the background chapter, supports the development of Adaptive SSL and ensures its practical utility. Adaptive SSL is shown to be effective in supporting runtime adaptation with minimum overhead and has been instrumental in illustrating our approach to achieve adaptive security for non-adaptive systems.

## 1.6 Outline

This chapter introduced motivations and goals for an adaptive security solution as well as detailing key thesis contributions. We now outline the primary thesis chapters:

**Chapter 2** Adaptive Security covers a wide range of research areas/domains with little shared common ground. As such we first present and discuss current Adaptive Security literature in this background chapter. This provides the context for and nomenclature to describe our own work.

**Chapter 3** We focus on our security domain and undertake an in depth comparative evaluation of current cryptographic algorithm implementations.

**Chapter 4** A methodology to build a runtime adaptive security solution is detailed in this chapter. We also consider the scope of such a methodology as well as providing a design for the new adaptive security solution.

**Chapter 5** In this chapter we detail and evaluate our Adaptive SSL solution which facilitates runtime security adaptation between client and server based on the design discussed in the previous chapter.

**Chapter 6** We study the effects of external factors influencing server related security cost. Insight gained through this offline study aides in formulating an adaptive security policy and we demonstrate its effectiveness in trading off security and performance at runtime.

**Chapter 7** Lastly we summarise and recapitulate the main themes of the thesis. We also consider possible future directions.

# Chapter 2

# Background

We conclude from Chapter 1 that adaptive systems typically require additional information, not traditionally accepted as a justified concern of the system, to make adaptation decisions. Such a general view of adaptive systems lead to a research area which is vast, multidisciplinary and involves a wide range of systems [17]. Not only is work related to this area far reaching but also relatively new. Almost all papers cited in this chapter were published this side of 2000, if not within the last few years. The combination of these factors contributes to the fact that current research in one area of adaptive systems is often ill suited for transfer to another. With regards to this, an adaptive systems Road Map paper published in 2008 states that "finding a solution that should be able to fit all the purposes might be remote." [17].

In this chapter we therefore take on the challenge of exploring the diversity of this exciting new area and endeavor to draw together some core principles on which these diverse adaptive systems are based. We examine what can be implied by the term adaptive and how it is achieved. We furthermore present current Adaptive Security literature providing a context and motivation for our research.

In the following section we firstly explore what *Adaptive* means and how it is utilised in adaptive security research. We do this by bringing current adaptive security research together under one adaptation taxonomy. Given an understanding of how adaptivity is achieved in its many forms we then focus on the subset of systems more closely resembling our research, namely *Self-Adaptive* systems.

## 2.1 Adaptive

In this section we endeavour to provide a more concrete understanding of the ambiguous term *adaptive* and draw together adaptive security research as they relate to different kinds of adaptivity. In doing so we note that bringing together adaptive security research in this fashion is to the best of our knowledge a unique compilation.

Using the adaptation taxonomy by S. M. Sadjadi et al. in [18, 19, 20] as a framework, this Chapter brings together Adaptive Security research by exploring how, when and where adaptation is utilised. For each section we discuss adaptive security research which best demonstrates the related concept, noting that such research may also be suited to one or more of the other sections.

### 2.1.1 When to adapt

Differences in adaptation techniques can be seen as a function of time [18]. Generally speaking, techniques which allow adaptation later in the product life cycle are much more powerful. Later adaptation does however mean that it is difficult to employ traditional testing and formal verification techniques to ensure system consistency. We first look at static adaptation early in the product life cycle and then move to more dynamic types of adaptation later in the life cycle.

#### 2.1.1.1 Static adaptation

Adaptive behavior which is *hardwired* into the system at development time cannot be changed without recoding. This clearly limits the ease with which the system can evolve over time.

Alternately the system can support *customization* at compile/link time. This is typically done to make an application suitable for a particular environment during deployment. It can be seen in traditional Linux operating systems where application source code is compiled/linked using a makefile script to make it suitable for that particular Linux distribution. Aspect Oriented Programming (AOP) languages also allow blocks of code to be inserted at various predefined points in the system during compile/link time. This code may deal with other system concerns such as security,

logging, etc. (see Section 2.1.2.2 for full description). These systems only require recompilation or relinking to be made suitable for a new environment.

*Configurable* systems delay the decision on which components to use for a particular task until the component is needed. For example the Java Virtual Machine (JVM) loads classes when the application first needs them. Hashii et al. [21] applies this to the security domain by utilising Java's Dynamic Classes to support reconfigurable security policies for mobile programs. Such programs traverse many different systems and so would benefit greatly from the ability to adapt appropriately to each new environment.

#### 2.1.1.2 Dynamic adaptation

Both *tunable* and *mutable* application types initiate adaptation at runtime. Systems which do not change their functional code at runtime but allow other crosscutting concerns to be altered in response to current environmental conditions are considered *tunable*. Examples include a paper by Brenda Timmerman [22] which considers the issue of dynamically masking network traffic to protect against traffic analysis attacks. It allows the cross-cutting concern of security (i.e. the amount of masking) to be altered in response to changing security policies, which in turn is partly based on current system load. Higher network load might result in a policy change to reduce network masking and so free resources to deal with the increased network traffic, and vice versa. Lawrence Teo et al. [23] describes a dynamic risk-aware access control architecture which provides additional runtime support to firewalls by monitoring the environment. It monitors client requests at runtime and makes intuitive risk based assessments on each request before allowing traffic through. Similarly R.M Venkatesan et al. [8] considers a firewall which adapts to threats by changing security policies for each user at runtime based on Intrusion Detection System (IDS) input. Kang et al. [24] also considers using IDS input in the context of real-time embedded systems to allow the system to perform optimally until a real security threat occurs. Finally M.E El-Hennawy et al. [25] also tries to alleviate security processing costs through segmenting data and applying a different level of encryption to each segment by varying the algorithm key size.

*Mutable* systems additionally allow adaptation which dynamically alters the system's behavior at runtime to one that is functionally different. As mentioned above this is a very powerful adaptation type and is typically constrained to avoid anomalous system behavior across adaptations. Chunxiao Chigan et al. [10] presents one such mutable security service for Mobile Ad-Hoc Networks. Not only can it change the level of security provided but allows complete changes to security infrastructure in response to different types of security threat.

The Adaptive Security solution presented in this thesis is a *dynamic tunable* solution as it enables adaptation of crosscutting concerns in response to current environmental conditions. More precisely, it enables runtime security adaptation based on current server load. This is achieved through a policy based mechanism which provides better security when resources are available whilst still respecting client Quality of Service constraints.

### 2.1.2 How to adapt

This dimension of the adaptation taxonomy reflects on the techniques used to support adaptation.

#### 2.1.2.1 Parametrisation

*Parametrisation* supports adaptation through a tunable variable whose value reflects either a user choice or environmental property. Such systems typically support static adaptation where the adaptation logic is hardwired into the system.

Generic Authorisation and Access Control (GAA-API) [26] is one such security system that supports adaptive authorisation through a tunable system threat level parameter. Policies are dynamically chosen and applied based on this parameter.

#### 2.1.2.2 Separation of concerns

Adaptation logic which is dispersed throughout the system is costly and difficult to modify and maintain [27]. Techniques which support *separation of concerns* address this problem and significantly simplify the development process. One such technique

to separate the concerns of functional behavior, i.e. business logic, with other crosscutting concerns such as security, Quality of Service and fault tolerance is Aspect Oriented Programming (AOP) [28]. This separation is achieved at development time by defining 'pointcuts' in the code where the code for crosscutting concerns (or 'aspects') will be inserted at compile or runtime using an 'aspect weaver'. This technique supports adaptation by firstly separating the various concerns and so allowing one 'aspect' to be easily replaced with another at compile time (i.e. customisable static adaptation. See Section 2.1.1). Secondly it supports tunable dynamic adaptation by allowing aspects to be weaved in at runtime [29, 30]. Lastly it can also support mutable adaptation by considering the adaptation itself as an aspect and so allow the system to adapt its functionality to that which was perhaps not envisioned during development time [31, 32]. This also requires reflection to some extent and is discussed in the next section.

AOP is still however considered relatively new with some potential unresolved issues [33, 6, 34, 35]. None the less Abdelkarim Erradi et al. [36] have taken these ideas and developed AdaptiveBPEL, a policy-driven framework for adaptive web services composition. Leveraging AOP, a mutable security framework for distributed object middleware was developed by Ruibing Hao et al. [37].

Our Adaptive Security solution upholds the idea of separation of concerns by distinctly separating the concerns of security from that of server configuration. This separation allows security infrastructure to be developed, changed and deployed independently and parallel to that of the server.

#### 2.1.2.3 Computational Reflection

*Computational Reflection* aids adaptation by allowing an application to reason about and alter its own structure and behavior. This is achieved through 'introspection' which allows the application to observe its own behavior and 'intersession' which enables the application to adapt its behaviour based on the observed behavior. Both are achieved through exposing selected details of the underlying system at a level of abstraction that hides unnecessary details whilst still allowing changes to the system behavior. The RUNtime Extension of Services (RUNES) middleware [38] architecture

utilises reflection, amongst other techniques, to provide a reconfigurable component based architecture for networked embedded systems.

#### 2.1.2.4 Component based design

*Component based design* (CBD) views the system as a group of components which conform to a set of well defined interface specifications. Using interfaces to standardise interactions between components allows $3^{rd}$ parties to independently develop software components for later integration with the system. These are also known as "Commodity-off-the-shelf" (COTS) components as they can be ready-made for use in any system that adheres to the interface specifications. Components facilitate adaptation either through static recomposition, where components are selected at compile time, or dynamic recomposition, where new components can be bound to the system at runtime.

Our Adaptive Security solution fosters the notion of CBD by enabling security adaptation through a standard HTTP based interface. This allows any $3^{rd}$ party which adheres to the interface to adapt the security.

### 2.1.3 Where to adapt

The last dimension of the taxonomy focuses on where in the system the adaptation is realised. In particular, we focus on Application, Middleware and Operating System layers.

#### 2.1.3.1 Application Layer

Adaptive code at this level provides adaptive support to programs which are not generally tied to a particular middleware platform. The following papers address various security issues at the application layer by providing infrastructure support through leveraging techniques presented in Section 2.1.2.

The Willow Architecture [39] supports fault and intrusion tolerance for critical distributed applications. It does this through a combination of component based design and control loop support. In the former distributed services are considered components which adhere to a specific API and can thus be dynamically exchanged

for other services at runtime and the latter provides support to monitor, diagnose and respond to application state changes.

The security API for the Strata software platform [40] allows dynamic adaptation of security policies at runtime based on system events. Their Aspect Oriented approach allows authorisation policies to be dynamically weaved into the application code at runtime.

Leveraging reflection B. Hashii et al. [21] developed an application layer Extensible Security Infrastructure to support dynamic access control policies.

Adaptive Trust Negotiation and Access Control (ATNAC) [41] supports security adaptation through parameterising their framework with threat-level and suspicion-level parameters. These parameters are used by the Generic Authorisation and Access Control API (GAA-API) [26] and TrustBuilder [42] to provide an adaptive access control and trust negotiation framework.

Through the use of Component Based Design and the concept of separation of concerns the Analyse and Plan components of the control loop for our Adaptive Security solution is realised at the Application level. Server load data is analysed, adaptation policies applied and appropriate instructions formulated to trade off security and performance.

#### 2.1.3.2 Middleware

Middleware provides adaptive support just below the application layer. Schmidt [43] further breaks this layer down into four layered sections as discussed below starting with the lowest middleware layer and moving up:

The *Host-Infrastructure* layer encapsulates and enhances lower level communication and concurrency mechanisms to hide the heterogeneity of the Operating System and hardware. This service is exposed through an API to the higher levels. Examples of adaptive middleware at this layer include ACE [44], Rocks [45] and MetaSockets [46].

The *Distribution* layer provides a programming abstraction to the layers above, allowing distributed applications to be written in a similar fashion to stand alone programs. That is, function calls on remote objects are free from hard coded depen-

dencies on their location, target programming language, target OS, etc. Adaptive middleware at this layer includes TAO [47], DynamicTAO [48] and OpenORB [49].

*Common Services* provide high level domain independent components that provide common services such as load balancing, logging, security, fault tolerance, real-time scheduling, etc. QuO [50], IRL [51] and FRIENDS [52] are examples of adaptive middleware at this layer. The domain independent Monitor and Execution components of our Adaptive Security solution are implemented at this level. A runtime adaptive Secure Socket Layer (SSL) service adapts a subset of client SSL sessions in response to application level requests. Current system load is also made available to the application layer.

Finally, the *Domain-Services* top layer is tailored to the particular domain and so can only be reused for that domain. Boeing Bold Stroke [53] open architecture for mission-computing avionics capabilities is an example of middleware at this layer.

#### 2.1.3.3 Operating Systems

Operating Systems (OS) is the lowest layer we consider. This is a vast field of research which we will not attempt to cover comprehensively. We do however take this opportunity to make note of particular adaptive security research in these areas.

H. Hinton et al. [12] developed a Security Adaptation Manager that adaptively protects the OS against buffer overflow based stack-smashing attacks. Complete copies of the system, each compiled with different levels of security in mind, are maintained. The system initially operates in a less secure/more performant state. System events are monitored and once an attack or sequence of potential attacks are observed future client requests are diverted to a more secure/less performant compiled implementation. Singh et al. [54] investigates the security-performance contention in the Access Control mechanism of Storage Area Network file systems. A compromise is found by developing trust/distrust in users based on their behaviour over time. Trustworthy users only require minimal security checks and so free up system resources.

## 2.2 Self-Adaptive

Having gained a deeper understanding of how adaptivity is achieved in current adaptive security literature we now further analyse those security systems which are considered to be *self-adaptive*. We first consider self-adaptive systems more generally, noting what current research considers as key key components of such systems. In section 2.2.1 we then analyse current adaptive security systems in that light.

A distinguishing characteristic of self-adaptive systems relates to how the adaptation is initiated and in particular by whom. One might think of this entity as the 'composer' who uses these techniques to adapt the system. The composer might be human, a system administrator or developer, or a piece of software. Software systems can be considered on a sliding scale from manual (human composers) to fully autonomic systems (software composers). The first level represents a system which requires skilled professionals to install, monitor, manage and replace. In contrast, the other side of the sliding scale represents an autonomic system which can automatically take appropriate actions based on business policies and objectives using available internal or external information [55]. An analogy for such systems can be found in biological systems such as the human nervous system which frees our brain from the burden of dealing with lower level, but still vital, functions such as heart rate and body temperature [56]. Autonomic systems are also commonly known as Self-star or Self-* [57] systems and can be categorised under four general Self-CHOP [58] characteristics (namely Self-Configuring, Self-Healing, Self-Optimizing and Self-Protecting [56]).

Such Self-Adaptive systems can be static or dynamic (see Section 2.1.1) in terms of when adaptation occurs but as autonomicity increases so does the need for a software representation of the decision making process. This process can be seen as a feedback- or control loop. It allows systems to make their own adaptation decisions without human intervention.

The generic autonomic feedback loop considers all stages of the decision making process [59]. The cycle starts with the collection of relevant environmental data. Issues such as the cost and frequency of collection should be addressed here as monitor-

ing itself could overload the system or negate performance benefits achieved through adaptation [17]. Collected data reflects the current state of the system and is analysed in the next stage. The analytical stage deals with modelling the current and possibly future state of the system and applying various rules and theories to the collected data. In the next stage a decision is made, potentially amongst alternatives presented in the previous stage using techniques such as risk analysis or decision theory, to reach a desirable state. This stage could also consider the impact of each alternative on other components sharing the same resources, also bearing in mind the duration and overhead of the adaptation process itself [17]. Finally the appropriate adaptation decision is acted on. The impact data of the adaptation can then be collected and used as feedback for the next control cycle.

Even though such feedback- or control loops have had much success in different branches of engineering, such as control theory [60], it is still unclear whether general principles of this discipline can be applied directly to self-adaptive software systems [17]. Control theory deals well with closed systems whose components and properties are well known and described using various linear or nonlinear mathematical models. More general systems however (e.g. discrete and continuous, time-varying, having delayed or uncertain information) whose structure is also perhaps not fully known is problematic in control theory even if they can be characterised mathematically [59].

Not only is control theory somewhat limited but the feedback loop is often an emergent property of the system rather than being explicitly represented in the system structure. In particular, autonomic systems can be organised into two groups, many adaptive systems sharing aspects of both. Nomenclature has not yet been standardised and so groups can either be top down/bottom up, weak(centralised)/ strong(decentralised)[17], tightly coupled/decoupled [61], etc. For the former the system is viewed in a top down fashion, most likely having some central point of control maintaining an explicit representation of the system and making adaptation decisions based on some higher level business goals. Such systems are more likely to suffer from scalability problems [17]. The latter is a system viewed in a bottom up manner where there is no central control but each component has a set of duties which, when viewing all components as a whole, produces some emergent behaviour. As such it can be

considered more difficult to predict and verify this emergent behaviour. It may also be an indication why many adaptive software system designers acknowledge the feedback properties of such systems but do not represent these control loops explicitly in design documents. Control loops are also often hidden in design documents through various abstractions intended to hide complexity. These give rise to a lack of visibility which can cause designers to overlook key control aspects and makes software validation and verification increasingly difficult [62].

In a bid to deal with the complexities of modern day computing systems IBM has taken the above challenges to heart and developed an "Architectural Blueprint for Autonomic Computing" [63]. At the core of this blueprint is the 'autonomic manager' which can be seen as a derivative of the popular engineering Model Reference Adaptive Control (MRAC) [60] control loop. The autonomic manager is an architecture that implements the generic feedback loop. First the Monitor component collects appropriate data from the managed resources through sensors. Data is correlated, filtered and/or aggregated and the discovered symptom is passed to the Analyse component. Symptoms and other data may also be stored in a shared knowledge base. The analyser determines whether a change needs to be made based on the shared knowledge (potentially a policy) and the symptoms. A change request is passed to the Plan component if necessary. The planner generates the necessary command or workflows in the form of a change plan which is passed to the Execute component. The executor performs the change plan on the manage resource using the effectors. The knowledge base may be updated if necessary.

Thus far only a small step has been taken to make feedback loops more visible and explicit in adaptive software systems [62] and it would be a worthwhile endeavor to further mine the rich area of control engineering to further ground adaptive systems research [17].

In recognition of rapidly changing environments and the need for explicit feedback loop representation L. Marcus [13] introduces the concept of an Adaptive Security Infrastructure (ASI). It is considered to be a hierarchical structure consisting of many autonomic systems. The paper focus' on the foundations for local and global policy specification in such an infrastructure.

Alexander Shnitko [14] combines research in [13, 64, 65] to produce a general model for an adaptive Complex Information System (CIS). The model also has clear similarities to a feedback loop or autonomic manager. Lastly, Carlo Montangero et al. [66] discusses an approach to the logic specification of an ASI in the secure mobile networking domain using ΔDSTL(x) spatial-temporal logic. The contributions in this thesis fit within this class of systems but we will discuss that in greater length in Section 2.2.2.

Motivated by the the fact that the feedback loop mechanism is often not explicitly considered when designing self-adaptive systems we analyse current adaptive security literature with respect to the IBM's autonomic manager in Section 2.2.1. This is followed by a discussion of our contribution in light of these findings in Section 2.2.2.

### 2.2.1 Feedback loop

In order to bound the discussion we now focus on adaptive security research in particular. Specifically, we address the feedback loop properties of current adaptive security systems in context of IBM's autonomic manager in this section. Each component of the autonomic manager is considered in turn, discussing if and how current adaptive security systems account for these components.

A summary of our findings is presented in Table 2.1. Each component is accounted for and an additional column, namely "feedback", is added to explicitly address the fact that the impact of the adaptation on the security system serves as feedback for the next control loop cycle. In other words, when security is adapted through the Execute component the Monitor component's measurements are affected in the next feedback loop cycle because of the adaptation. With reference to Table 2.1 we now explore how current adaptive security systems address each component in turn. These papers are chosen as they most closely represent our background discussion as well as the various aspects of our research contributions.

#### Execute

The Execute component represents the security adaptation event itself and as such we find that all Adaptive Security systems minimally support at least this part of the

Table 2.1: Feedback loop components

| Paper | Execute | Monitor | Analyse/Plan | Feedback |
|---|---|---|---|---|
| Adaptive SSL [67] | Yes | Active | Offline | Yes |
| Authenticast [11] | Yes | Active | Online | Yes |
| Sang H. Son et al. [68] | Yes | Active | Online | Yes |
| B. Timmerman [22] | Yes | Active | Online | Yes |
| Kang et al. [24] | Yes | Active | Offline | Yes |
| C. Chigan et al. [10] | Yes | Passive | Offline | |
| AEF [23] | Yes | Passive | Online | |
| EAC [8] | Yes | Active | Offline | |
| SAM [12] | Yes | Passive | Offline | |
| Sigh et al. [54] | Yes | Passive | Online | |
| ADME/AF schema [7] | Yes | Passive | Online | |
| El-Hennawy et al. [25] | Yes | | | |

feedback loop. Even the "Adaptive Security/Performace encryption system" by El-Hennawy et al. [25] addresses this core need when adapting between different key sizes for the encryption of single files. Even though they do consider the performance cost/benefits of such an adaptation they do not monitor its effect on the current system performance, nor make adaptation decisions based on such insights.

From Table 2.1 one can clearly see that the Execute component is not an overlooked consideration when creating Adaptive Security systems and is a key component in enabling security adaptation in an Adaptive Security solution.

### Monitor

With regard to the runtime monitoring of the system state we found that there were two types of monitoring taking place.

The first can be classified as *Passive* monitoring. The system waits for an event to occur. Adaptive access control systems are particularly suited to this type of monitoring as the monitoring system must wait for a user to access the system. L. Teo et al. created an Authorisation Enforcement Facility (AEF [23]) for network access management which provides passive monitoring by scanning incoming packets for suspicious content or behaviour. Sigh et al. [54] also uses passive monitoring when building trust/distrust in users when they access the Storage Area Network file system. Lastly, K. Beznosov proposed ADME/AF schema [7] to enable application specific access control in middleware. Passive monitoring comes into affect when the

application and middleware layer interact.

The second is *Active* monitoring and involves the monitoring system proactively polling to retrieve sufficient data. This can most clearly be seen in adaptive security systems where the system resources such as load or performance is monitored. The Authenticast system [11] actively monitors system load to provide dynamic authentication for streaming services.

The Enhanced Access Control system (EAC [8]) makes use of both techniques to detect threats. It actively analyses audit trail data and passively monitors user activity when they connect to the firewall.

We note that Active monitoring is significantly more resource intensive and as such could negate the performance benefits achieved through adaptation [17]. Attributes affecting such costs such as the frequency of monitoring and cost of calculating performance averages are not specifically addressed by the above papers. Monitoring provides current runtime information to aid in security adaptation decisions and so forms another key component in the feedback loop.

**Analyse/Plan**

At the core of the decision process is the Analyse component. According IBM's Autonomic blueprint [63] all components need not always be present and since the output of the Analyse component is often a adaptation decision for the Execution component we consider the Analyse and Plan components together.

As the analysis can be very system specific, this component can be realised in a myriad of different ways. We make the following observations regarding trends in this area.

Firstly, even though the analysis occurs at runtime a number of systems require additional *offline* measurements or computations to aid in the runtime (i.e. online) analysis. Such decision aids need to be pre-computed offline before the system can be utilised. C. Chigan et al.'s paper on "Resource-aware Self-Adaptive Security Provisioning in Ad Hoc Networks" [10] introduces an offline optimal secure protocol selection module to analyse and select the most cost effective stack of security protocols from various layers of the OSI stack to be utilised together at runtime. This selection

is a costly process which may cause performance degradation in the running system and is thus done offline. The Enhanced Access Control (EAC [8]) system makes use of offline audit trails to learn what can be classed as normal user behaviour. Rules derived from this processor intensive process is then used at runtime in EAC Firewall rules. Lastly, the Security Adaptation Manager [12] can switch to a more secure version of the system when a threat is detected. Multiple versions of the system is compiled and maintained offline, each with different security charateristics, enabling the system to choose between them at runtime. In all the papers above we see that the analysis is a resource intensive activity and as such best achieved offline.

Secondly we note that compared to offline analysis the complexity of the *online* analysis process is limited due to time constraints and processing resource limitations. The Authorisation Enforcement Facility (AEF [23]) uses a simple threshold value denoting initial trust in the user together with threat level data from the monitor component to make runtime security adaptation decisions. Sang H. Son et al. in their paper on "An Adaptable Security Manager for Real-Time Transactions" [68] consider the cryptographic security of real-time database transactions. They developed an Adaptable Security Manager which utilises a simple feedback mechanism to link transaction deadline completions, as reported by the monitoring component, with a security level. If too many deadlines are missed a percentage of transactions are moved to a lower security level to free resources, and vice versa. Authenticast [11] also links security level with available resources through a feedback mechanism. In this case authentication of multimedia streams is considered. If the rate of authentication falls below the stream arrival rate, security is adapted through a set of heuristics to reduce resource consumption and so allow the equilibrium between arrival rate and authentication to be restored. Heuristics include only authenticating a percentage of the incoming stream, delaying authentication and changing the security algorithm.

We also note that the above mentioned papers merely report on the performance cost savings of using heuristics, percentage based decisions and threshold values but do not consider the performance effect of such decisions as part of the decision process. To make an informed decision and effectively decide whether security should be adapted one must know the performance effect of adapting the security on the system

before doing so. Taking this into consideration may however be too computationally expensive for online analysis. Offline analysis may also find this difficult due to the online nature of the adaptivity and its performance effect.

### Feedback

The final aspect of Self-Adaptive systems to consider is the notion that changing the security has an impact on the environment which in turn serves as feedback to the next iteration of the feedback loop. We found that this is not always the case as the adaptive security system may monitor a resource which is not affected by the security adaptation. This is for example the case for the Security Adaptation Manager [12] which adapts to a more secure version of the software when a security threat is detected. The threats are monitored and reacted to in the same way in this feedback loop as in the next iteration of the loop.

Kang et al. in their paper entitled "Towards security and QoS optimization in real-time embedded systems" [24] the system performance is affected when a level of security is selected to encrypt the transmission. This in turn will have a direct impact on future decisions as the system is now in a less/more performant state and more/less capable of meeting the real-time deadlines.

Authenticast [11] is another such system where adapting the security (i.e. the authentication of real time data streams) directly affects the future system performance. As such different decisions may need to be made on the next iteration of the feedback loop.

B. Timmerman in her paper "A security model for dynamic adaptive traffic masking" [22] addresses the issue of adaptive network traffic masking to protect against traffic analysis attacks. The amount of masking is sensitive to the current network load and also affects the network load directly when the amount of masking is adapted. Changes in the masking frequency therefore has a direct impact on the adaptation decision for the next feedback loop iteration.

Lastly we consider Sang H. Son et al.'s paper on "An Adaptable Security Manager for Real-Time Transactions" [68] where the cryptographic security of real-time database transactions is addressed. Transaction deadline completions are measured

and a percentage of the transactions are adapted to a lower security level if deadlines are missed, and vice versa. Adapting security affects system performance and therefore by implication how many deadlines are met. The impact caused by the security adaptation therefore provides feedback for the next feedback loop iteration.

We feel it is important to address this aspect of the feedback process as it is often overlooked in current literature. It also adds significant additional complexity to the Analysis/Plan components and is therefore an interesting research area to explore. When considering the papers in this area we also note that even though the adaptation has a direct effect on the environmental attributes that are monitored by the Monitor component in the next iteration of the feedback loop, the adaptation itself does not explicitly take the future state of the system into account when making such decisions in the current iteration of the feedback loop. Amongst other things we address this in our contribution discussion next.

### 2.2.2 Contribution discussion

In this thesis we take the ideas discussed in this chapter one step further to create a methodology capable of extending standard legacy security systems, making them fully self-adaptive. The methodology provides steps and considerations to address all the components of the autonomic feedback loop and in so doing move adaptation decisions for existing non-adaptive security systems from design or deployment time to runtime. In particular we advocate a fully Self-Adaptive system where the security adaptation has a direct affect on the monitored resource (i.e. The impact caused by the security adaptation provides feedback for the next feedback loop iteration.). To study the interrelationship between security and the monitored resource we introduce an offline element to the Analysis/Plan component. As we will see, studying this relationship through an offline measurement based approach equips us with the tools to predict the system performance behaviour resulting from a security adaptation. This aids greatly in developing intelligent and effective adaptation policies. We note that the type of monitoring to be used is not stipulated in the methodology.

In this thesis we apply our methodology to the SSL/TLS security protocol [69]. In the first row of Table 2.1 one can see that our new Adaptive SSL (ASSL) system

has an Active monitor component (i.e. processing resources is monitored), an offline element to the Analyse/Plan component (from which our security-performance trade-off policies are devised) and exhibits the aforementioned 'Feedback' property. The offline element of the Analyse/Plan component is of particular interest in this implementation. Through offline experimentation we measure the system states under different security algorithms and in so doing enable us to predict the future system performance state resulting from a security adaptation. This results in intelligent and effective adaptation policies. In context of the taxonomy in Section 2.1 our ASSL solution can be described as a dynamic tunable solution. Concepts such as Component Based Design and Separation of Concerns aided in developing the Monitor, Analyse, Plan and Execute components of the feedback loop which themselves are distributed over the Middleware and Application domain layers.

# Chapter 3

# Cryptographic Algorithm Implementations

In aid of exploring the contention between security and performance we investigate the cryptographic algorithms required to secure client-server interactions. We study both the level of security they provide and the performance cost of doing so.

Algorithms typically provide the following security guarantees: message integrity to ensure messages are unaltered during transit; message confidentiality to ensure message content remain secret; non-repudiation to ensure that the sending party cannot deny sending the received message; and authentication to prove sender and/or receiver identity.

We first introduce the cryptographic algorithms needed to achieve the above security guarantees and show how they are combined to provide a more effective security solution. We furthermore study current implementations thereof and show that performance overhead varies significantly amongst implementations.

## 3.1 Cryptographic algorithms

In this section we detail the most prevalent cryptographic algorithms used to achieve message integrity, confidentiality, non-repudiation and authentication. We also show how these algorithms are used together to build a more effective security solution.

### 3.1.1 Symmetric cryptography

Symmetric cryptography tries to ensure message confidentiality by encrypting the message (the plaintext) using a secret key to produce an encrypted version of the message (the cipher text), which is then sent instead of the original message [70]. Message integrity is implicitly provided, as altering the cipher text would result in an illegible decrypted message. Symmetric refers to the fact that the same secret key is required to decrypt the message on the recipient's side [71]. Typical symmetric encryption algorithms include DES, Triple DES, RC2, RC5, Twofish, Blowfish, IDEA and AES. Most symmetric algorithms can operate in different modes, most common of which are Cipher Block Chaining Mode (CBC) or Electronic Codebook Mode (ECB). The former is considered more secure as it ensures that encrypting the same plaintext never produces the same cipher text. An inherent problem in using symmetric cryptography is the key distribution problem; since the same secret key is used to decrypt the message, one must find a way to securely transport the key from sender to recipient.

### 3.1.2 Asymmetric cryptography

Asymmetric cryptography provides the same message security guarantees as symmetric cryptography, but additionally provides the non-repudiation guarantee. Asymmetric refers to the fact that different keys are used for encryption and decryption. One key is kept secret (private key) and the other is made public (public key), and both are unique. The recipient's public key should be used during the encryption process to ensure message confidentiality as only the recipient has the necessary secret key to decrypt the message. If, however, the message is encrypted using the sender's private key the sender cannot deny sending the message as his private key is unique and is only known to him. Typical asymmetric algorithms include RSA, ElGamal and DSA. Asymmetric cryptography is extremely powerful, but this comes at a cost. Especially for longer messages and keys, it is much slower than its symmetric cryptography counterparts [72].

### 3.1.3 Hashing

Hashing tries to ensure message integrity by producing a condensed version of the message, known as the message digest, which is unique to that message. The hashing algorithm is publicly known and so the recipient can perform the same hash on the received message, to produce another message digest, and compare it to the received digest to asses whether the original message has been altered. Typical hashing algorithms include MD2, MD4, MD5, RIPEMD, SHA-1, SHA-256, SHA-384 and SHA-512. Hashing does not provide confidentiality, non-repudiation or authentication. On its own, hashing does not provide message integrity either as both the hash and the message could be replaced by a $3^{rd}$ party and so prevent the recipient from detecting the attack. The next section explains how hashing is utilised to ensure message integrity.

### 3.1.4 Hybrid security system

The techniques detailed above are combined to achieve a more effective security solution through signing, verifying, encryption and decryption. They are combined as follows:

The key, in symmetric cryptography, can be securely transported using public key cryptography by encrypting the symmetric key using the receiver's public key. The receiver, and only the receiver, can then first decrypt the symmetric key using his private key and then decrypt the message using the decrypted symmetric key. Note that only the symmetric key, which is relatively short, is encrypted using public key cryptography thus reducing encryption overhead.

The message digest, produced by the hash function, can be encrypted using an asymmetric cryptography algorithm to avoid an interception attack. Thus, if the message digest is encrypted using the sender's private key, only the message can be replaced during transit and not the message digest, since the interceptor does not have the sender's private key to encrypt the new message digest.

Generating a message digest and then encrypting the message digest using a private key is referred to as signing the message. Decrypting the message digest using the

sender's public key, generating a new message digest of the received message and then comparing the digests is called verifying the message. The performance results of these two techniques, among others, are analysed in this chapter.

Sender authentication is achieved when the sender's public key is signed by a mutually trusted $3^{rd}$ party. The receiver can then verify the public key as the $3^{rd}$ party's public key is trusted.

#### 3.1.4.1 RSA

Understanding the security implications and performance results in Section 3.2 requires a deeper understanding of public key cryptography. In particular RSA [73], which was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977. We do not explain all the details of RSA, but instead focus on the particular use of RSA as detailed above.

**The algorithm** [74] [75]

- Choose 2 large primes $p$ and $q$ such that $pq = N$
- Select 2 integers $e$ and $d$ such that $ed = 1 \ mod \ \phi(N)$
  - Where $\phi(N) = (p\text{-}1)(q\text{-}1)$ is Euler's phi (or totient) function of $N$

$N$ is called the modulus, $e$ the public exponent and $d$ the private exponent. The public key is the pair ($N$, $e$) which is made public and the private key is the pair ($N$, $d$) which is kept secret. RSA encryption and decryption explained in context of the above sections is expressed as follows:

**Encryption:**

The symmetric key $M$:

Encrypted key = $M^e \ mod \ n$

The message digest $M$:

Encrypted digest = $M^d \ mod \ n$

**Decryption:**

The symmetric key $C$:

Decrypted key = $C^d \ mod \ n$

The message digest $C$:

Decrypted digest = $C^e \ mod \ n$

Where $M$ is the key or digest converted to an integer according to RFC3447 [76], $C$ the encrypted key or digest and $n$ the particular modulus, chosen to be either 512, 1024, 2048, 3072 or 4096 bits.

When studying performance, it should be noted that encrypting the key and encrypting the message digest is not the same function as one uses the public and the other the private exponent. Therefore, encrypting the symmetric key and decrypting the message digest (in the verification process) is mathematically equivalent as they both use the public exponent. The same can be said for encrypting the message digest (in the signing process) and decrypting the symmetric key as they both use the private exponent. RSA operation time greatly depends on the length of $e$ and $d$ [77], since longer exponents incur much larger time and therefore processing overhead.

In the following two chapters we consider how the length of the public and private exponents affect security as many security mechanisms exploit this to achieve faster symmetric key encryption/decryption and message signing/verification.

**Smaller public exponent**

The public exponent $e$ is used in symmetric key encryption and message verification. The smallest possible value for $e$ is 3 [78]. This can however weaken RSA confidentiality assertions. In particular, if the length $|M|$ of the message is such that $|M| < \sqrt[e]{N}$ the plaintext can easily be recovered [74]. Hastad's broadcast attack can be mounted if $k$ cipher texts, encrypted with the same public exponent, can be collected such that $k \geq e$. [78]. The Chinese Remainder Theorem (CRT) can then be used to recover the plaintext message [79, 78]. A defence against such attacks would be to pad the message using some random bits [80]. Coppersmith imposed further

restrictions on this in his Short Pad Attack which concludes that for $e = 3$ an attack can still be mounted, even though a random set of bits are used, if the pad length is less than $\frac{1}{9}$ of the message length [78]. The Public-Key Cryptography Standards RFC (RFC3447: PKCS♯1 [76]) does however propose the use of Optimal Asymmetric Encryption Padding (OAEP) [80] for new applications and PKCS1-v1_5 for backward compatibility with existing applications.

Although $e = 3$ can provide adequate security, if necessary precautions are taken, the current recommendation is $e = 2^{16} + 1$ [78] which is still small, requiring only 17 multiplications, but big enough to solve the above problems at the cost of a slight increase in encryption time. Short public exponents are not however a concern for signature schemes [79, 74].

**Smaller private exponent**

A shorter private exponent $d$ would result in faster key decryption and message signing. Typically the private exponent is the same length as the modulus regardless of the public exponent length. Wiener [81] has however shown that if $d < \frac{1}{3}N^{0.24}$ the private exponent can be obtained from the public key ($N$, $e$). More recently, Boneh and Durfree have shown this to be closer to $d < N^{0.292}$ [82, 75] and predicted the likely final result to be closer to $d < N^{0.5}$ [78, 82].

Other techniques used to decrease algorithm operation time include the use of the Chinese Remainder Theorem [78], know as RSA-CRT, which is said to be approximately 4 times faster than using standard RSA algorithms [75]. Rebalanced RSA-CRT can also be used and tries to shift the cost towards the usage of the public exponent $e$ [83, 81].

## 3.2 Performance analysis

This section studies the relative performance overhead between the different security algorithms discussed above and in particular shows how the choice of implementation can have a significant impact on their final performance cost. We first look at cryptography software on the client side and show how choosing either the standard Java Sun Cryptography Extensions [84] or the Java Cryptix Libraries [85] can significantly

influence the expected algorithm performance. Secondly we evaluate the supported algorithms on the server side (OpenSSL [86]) and show how these vary in performance not only compared to the client side implementations but also versions of the same software. Lastly we consider the security and performance of Verisign's TSIK toolkit which was recently acquired by the Apache Software Foundation [87]. It is a hybrid security system (see Section 3.1.4) used to facilitate Web Services security. As far as we are aware this is the only performance study of the TSIK toolkit.

We note that parts of the results are based on the MSc dissertation of P. Tomlinson [88], others are obtained by the author [89]. The results have been published as a joint publication [90]. To achieve a stand-alone discussion, these and additional results are presented in this chapter. In particular Figures in section 3.2.1 are by Tomlinson and all Figures in sections 3.2.2 and 3.2.3 are by the author.

### 3.2.1 Java Cryptography Extensions & Java Cryptix libraries

This section details a performance evaluation of the most common cryptographic algorithms for each cryptographic technique. Sun Java Cryptography Extensions [84] with libraries from Sun (referred to hereafter as JCE) as well as Java Cryptix Libraries for JCE [85] (referred to hereafter as Cryptix) are used for this purpose.

All experiments were conducted on a 1GHz machine with 256MB RAM running Linux Fedora Core. For each experiment a 1,137 byte plaintext file was used. All results for symmetric and asymmetric algorithms include key generation, algorithm initialisation and message encryption times. The experiments were repeated several times with negligible variance in the results.

The results presented suggest that RSA-1024 and SHA-256, with 1024 bit key size and 256 bit digest length respectively, are the most suitable cryptographic algorithms for use during transactions in systems with performance constraints. Almost any of the symmetric algorithms could be selected, but IDEA was shown to be the fastest in our evaluation.

#### 3.2.1.1 Symmetric cryptography

This section details a comparative performance evaluation of a subset of symmetric encryption algorithms. Using Cryptix we furthermore investigate whether either Cipher Block Chaining Mode (CBC) or Electronic Codebook Mode (ECB) boasts a performance advantage. 128 bit key size was used for all algorithms with the exception of DES (56 bits), Triple DES (112 bits) and Skipjack (80 bits) as they require fixed key sizes. Unless stated CBC mode was used.

### Algorithms

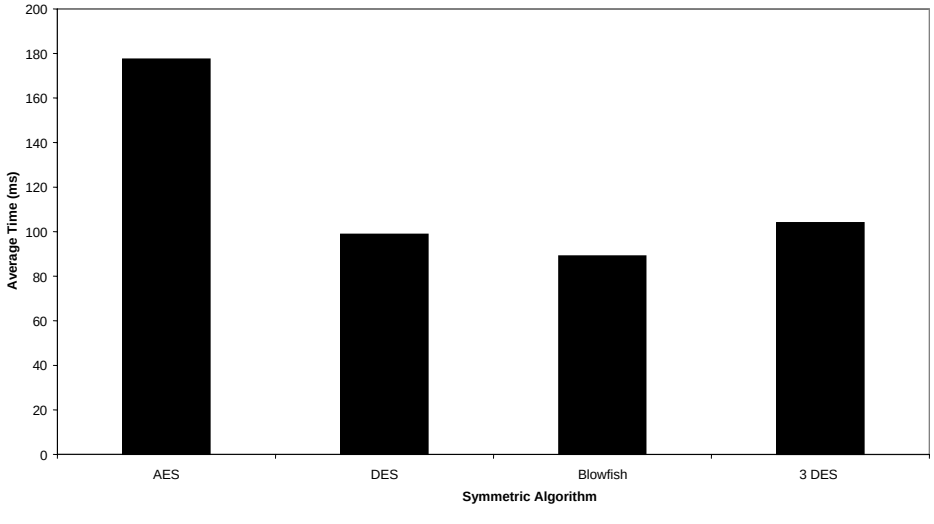


Figure 3.1: Average time to encrypt a 1137B file using JCE libraries from Sun

Looking at Figures 3.1 and 3.2, the first observation to make is that there are significant differences between the observed durations shown in each figure; JCE took much longer than Cryptix for the same algorithm. As an example consider the encryption time of ∼180ms for AES in JCE (Figure 3.1) and ∼40ms for AES in Cryptix (Figure 3.2). The implementation therefore has a large impact on the efficiency of the algorithm execution. This is further emphasized when individual

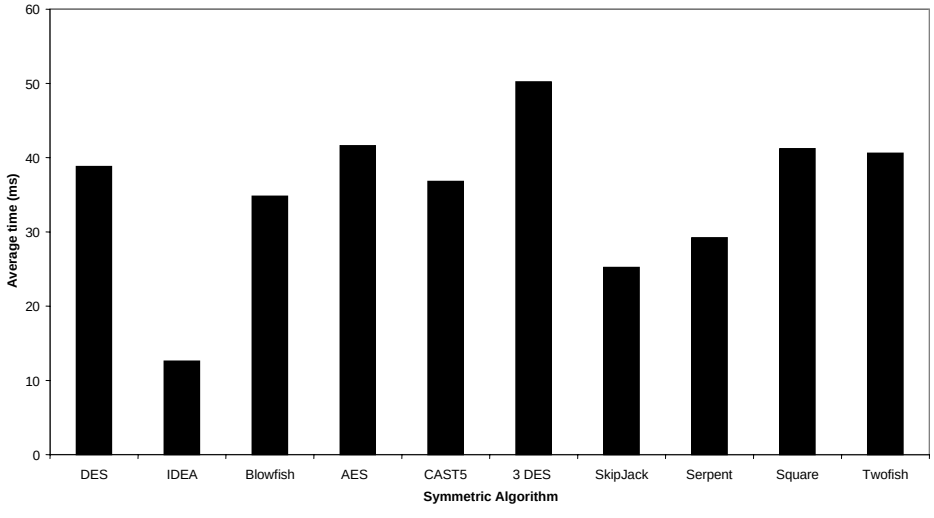


Figure 3.2: Average time to encrypt a 1137B file using Cryptix distributions

algorithms are compared. Naively one would expect that the 112 bit Triple DES (3 DES) would take twice as long as the 56 bit DES. However, this is evidently not the case, being only about 25% slower in Cryptix and only very marginally slower in JCE. Clearly this is influenced by the implementation, and conceivably Java Virtual Machine optimisations are also playing a part in apparently "speeding up" Triple DES.

The algorithm which performed the best in our evaluation was the International Data Encryption Algorithm (IDEA). According to Schneier [91], IDEA is approximately twice as fast as DES; in our experiments it was closer to three times as fast. Perhaps surprisingly, Blowfish was much slower, only a little better than DES and slower than algorithms such as Skipjack [92] and Serpent. Blowfish was designed to be fast and to require little memory [91], but we did not find this Cryptix distribution particularly efficient in our experimental set up. The Advanced Encryption Standard (AES) [93] performed particularly badly in the JCE distribution, but less poorly in the Cryptix distribution. We were unable to satisfactorily explain this difference,

except as further evidence of how the implementation of an algorithm can severely impact the actual performance.

What is not evident in these plots is the relative security of the different algorithms. In this respect key length is a good indicator, and so DES and Skipjack with key lengths of 56 and 80 bits respectively may be considered potentially less secure than others. Overall therefore it appears that IDEA is the best choice among the symmetric algorithms tested, as it provides adequate security as well as a fast execution time.

### Encryption Mode

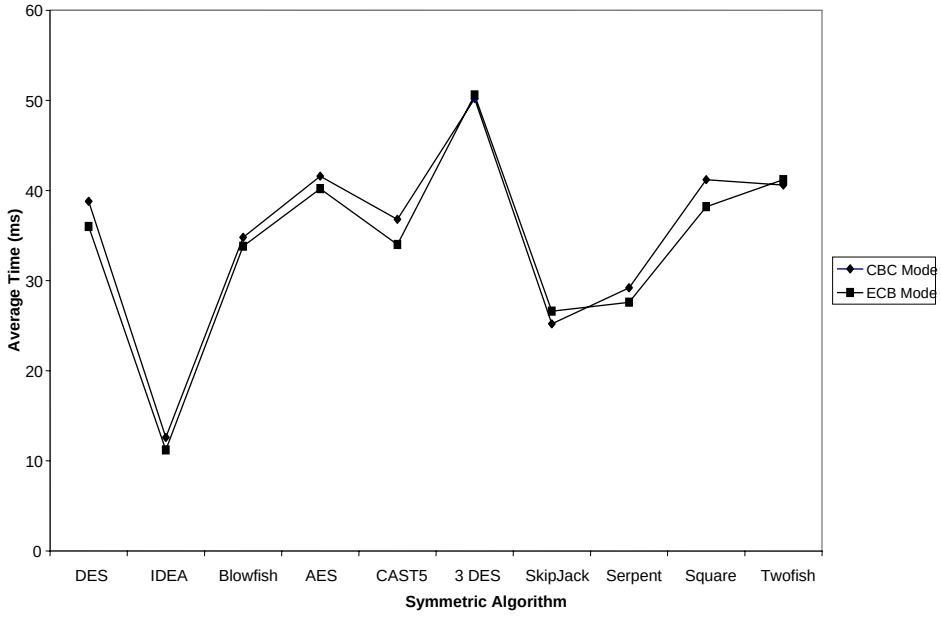


Figure 3.3: A comparison of symmetric algorithms operating in ECB mode and CBC mode

Figure 3.3 clearly indicates that neither mode shows a significant performance advantage. It would therefore seem prudent to use CBC mode during message encryption as discussed in Section 3.1.1.

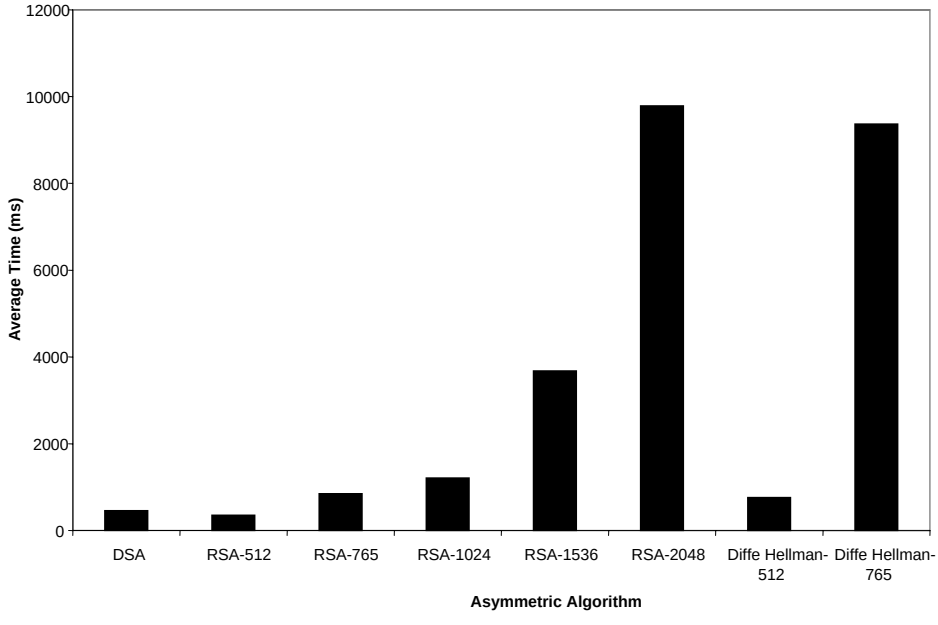


Figure 3.4: Average time to encrypt a 1137B file using public key algorithms

#### 3.2.1.2 Asymmetric cryptography

The results presented in Figure 3.4 were obtained using the standard Java Cryptography Extensions (JCE). The graph shows the average time to generate keys and process 1,137 bytes of data. Firstly we note that the increase in processing time is more than linear for RSA as key size increases. It can also be seen that the 1024 bit Digital Signature Algorithm (DSA) [93] outperforms both RSA-1024 and RSA-765. DSA is therefore a good option for signing data. DSA can however only be used for non-repudiation purposes whereas RSA can be used for both encryption and non-repudiation.

#### 3.2.1.3 Hashing

Java Cryptix Libraries were used in this experiment. All MD algorithms has a digest length of 128 bits, 160 bits for SHA (unless otherwise stated) and 192 bits for Tiger. As can be seen in Figure 3.5, SHA-1 significantly outperforms all other considered algorithms. Unfortunately SHA-1 has recently been shown to be less secure than

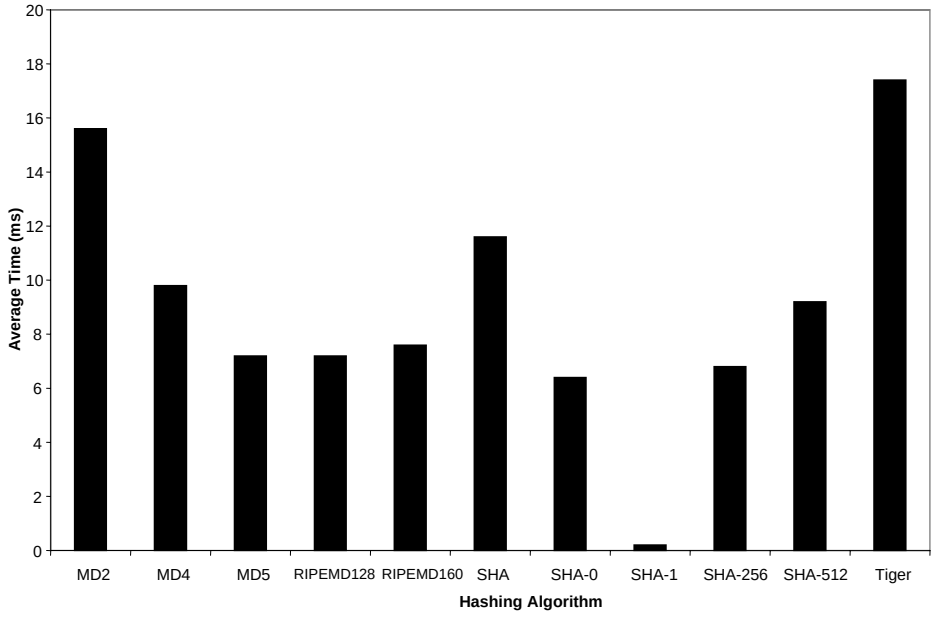


Figure 3.5: Average time to generate a message digest

initially anticipated and SHA-256 is currently recommended [94]. RACE Integrity Primitives Evaluation Message Digest (RIPEMD) with digest lengths of 128 and 160 bits have been developed to replace the 128 bit MD algorithms. Both RIPEMD algorithms seem to achieve comparable performance to that of SHA-256, though clearly have shorter digest lengths and so are potentially less secure.

#### 3.2.1.4 Summary

A comparative evaluation of the standard JCE libraries and Cryptix libraries for the main symmetric, asymmetric and hashing algorithms have shown that the chosen implementation plays a significant part in the expected performance of the particular algorithm. In particular, JCE performs worse than Cryptix for the same algorithm (see Figure 3.1 and 3.2). The symmetric encryption algorithm IDEA has been found to have particularly good performance and 3DES showed a performance cost not that much worse than DES (Figure 3.2). Clear choices for asymmetric encryption and hashing are RSA-1024 and SHA-256 respectively.

### 3.2.2 OpenSSL libraries

This section details our evaluation of algorithm implementations for each cryptographic algorithm supported on the server side. In particular we evaluate two versions of the OpenSSL libraries, namely 0.9.7f and 0.9.8d. We show that even amongst different versions of the same software algorithm performance can vary greatly.

All experiments were conducted on a Pentium III 866MHz machine with 512Mb RAM using the Linux "openssl speed" utility [95]. Figure 3.6 exhibit 90% confidence intervals of 60KB/s or less with the exception of 0.9.8d AES-128 and AES-192 with intervals of 400KB/s or less. Figure 3.7 shows 90% confidence intervals of 2KB/s or less, Figure 3.8 90% confidence intervals of 30KB/s or less and Figure 3.9 90% confidence intervals of 300KB/s or less.

We find that the particular implementation has a significant impact on the overall algorithm performance and so performance assumptions about known fast/slow algorithms can be misleading. Symmetric algorithm AES-128 and hashing algorithm MD4 performs particularly well for the newer 0.9.8d version whilst, compared to the client side Java implementations, SHA-1 performs poorly for both implementations.

We also note that algorithm performance results are measured in data throughput (kB/s) and not in overall processing time (ms) as in Section 3.2.1. When comparing these results with those in Section 3.2.1 we keep in mind that an increase in throughput indicates a faster algorithm implementation whereas an increase in processing time indicates a slower implementation.

#### 3.2.2.1 Symmetric cryptography

OpenSSL supports a wide variety of symmetric algorithms as can be seen in Figure 3.6.

Figure 3.6 shows that AES in particular has benefitted from a significant performance increase in the new 0.9.8d implementation. Where AES is one of the slowest implementations in the Cryptix and standard JCE implementations (Section 3.2.1) it is the fastest in OpenSSL, even surpassing the typically fast Blowfish algorithm. The relative algorithm performances of the older 0.9.7f implementation is however in

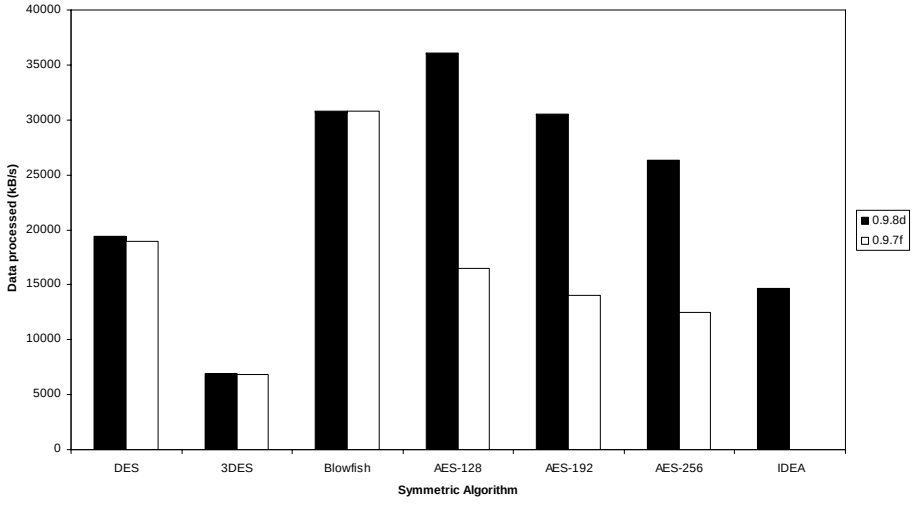


Figure 3.6: Comparison of OpenSSL symmetric algorithms

concurrence with Cryptix implementation. The figure also shows that an implementation of IDEA was introduced in the new 0.9.8d version, though not performing best overall as in the Cryptix implementation.

#### 3.2.2.2 Asymmetric cryptography

Figure 3.7 and Figure 3.8 show the signing and verification processes respectively (see Section 3.1.4) of the supported OpenSSL asymmetric algorithms.

Figure 3.7 shows that for the message signing process both algorithms suffered a slight performance decrease in the newer 0.9.8d implementation. DSA performs consistently better than RSA for message signing as key size increases from 512 to 2048, performing almost 20%, more than a 100% and almost 300% better. RSA is however still often chosen over DSA as it can be used in conjunction with any hashing algorithm and can also additionally be used to encrypt messages. These figures additionally affirm the results in Figure 3.4 where DSA-1024 outperformed RSA-1024 during the signing process for the standard JCE libraries.

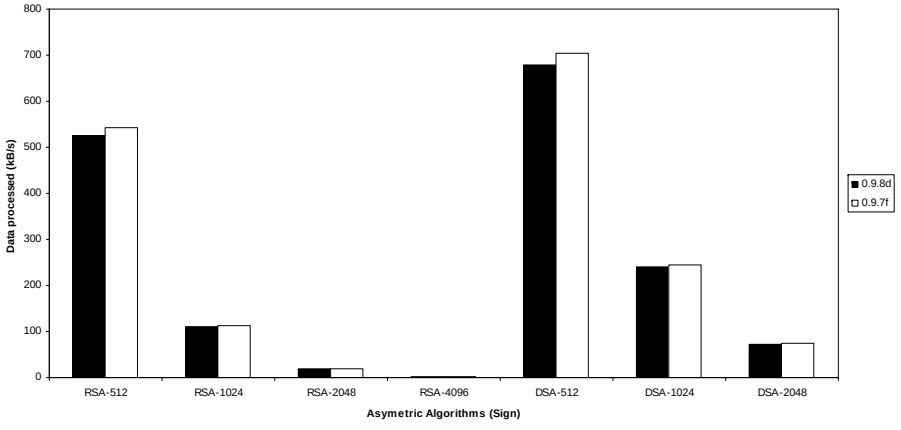


Figure 3.7: Comparison of OpenSSL asymmetric signing algorithms

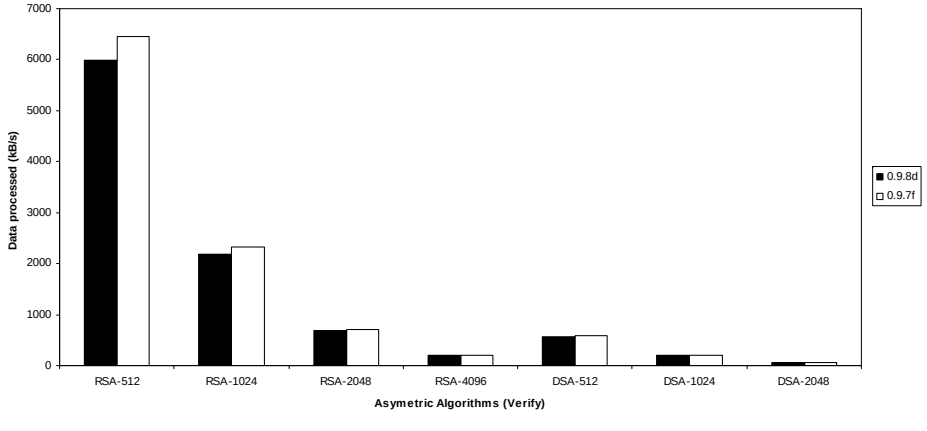


Figure 3.8: Comparison of OpenSSL asymmetric verification algorithms

The verification algorithms in Figure 3.8 show a slight performance decrease for both asymmetric algorithms in the newer 0.9.8d implementation. As is generally accepted, RSA performs consistently better than DSA during message verification.

#### 3.2.2.3 Hashing

Supported OpenSSL hashing algorithms are shown in Figure 3.9:

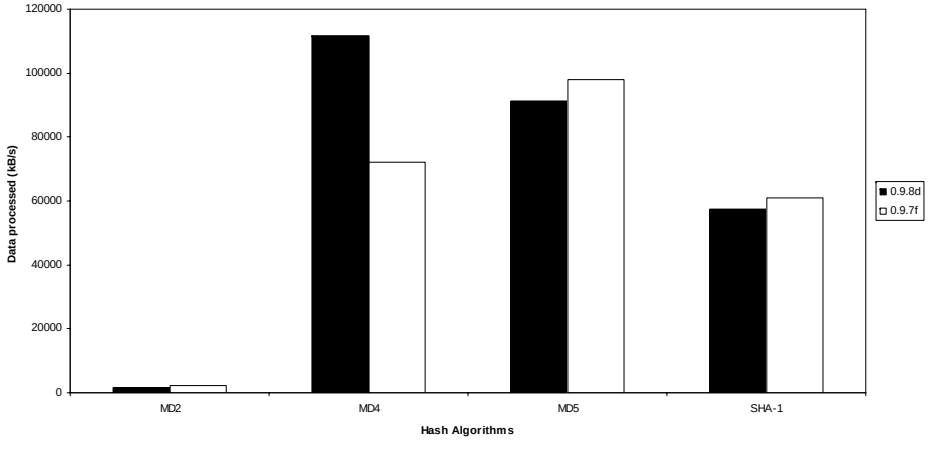


Figure 3.9: Comparison of OpenSSL hash algorithms

The figure shows mixed performance results for hashing algorithms in the new version. 0.9.7f shows performance improvements from MD2 through to MD5 but the new version shows MD4 performing better than MD5.

Looking at the Cryptix implementations in the previous chapter (Figure 3.5) we can see that SHA-1 outperforms the other algorithms by a large margin. In the OpenSSL implementation however it performs worse than MD4 and MD5.

#### 3.2.2.4 Summary

This section compared and contrasted two implementations of the OpenSSL cryptographic library with each other as well as with client side Java libraries discussed in the previous section. We have shown that even amongst minor software version alterations performance can vary considerably. In particular we have shown that when choosing a hashing algorithm the software implementation significantly impacts the performance amongst the MD algorithm family. OpenSSL's SHA-1 and IDEA implementations also perform poorly compared to the other OpenSSL algorithms. Implementations of these algorithms are however significantly faster than their peers in the Java Cryptix implementation. When considering the symmetric encryption algorithms, the newer OpenSSL 0.9.8d implementation provides a particularly fast version of the cryptographically strong AES-128 algorithm.

### 3.2.3 Web Services - Hybrid security system

In this section we consider the combination of cryptographic algorithms used in VeriSign's web service Trusted Services Integration Kit (TSIK) [87], currently part of the Apache Software Foundation [96], and evaluate them based on the level of security they provide as well as their performance. TSIK's performance is evaluated through direct comparison with Java's Cryptography Extensions (JCE). This is published in [90, 89].

We first analyse the level of security provided by the implementation of the hybrid system discussed in Section 3.1.4 and then evaluate its relative performance overhead through experimentation.

We found that TSIK typically has worse performance than JCE. Its performance is similar to JCE, except that it slows down when processing messages with large plaintext sizes. It also provides adequate confidentiality, non-repudiation and sender authentication guarantees through the use of Triple DES and RSA, though should consider using SHA-256 for message verification in future releases as suggested in recent literature [94]. With respect to technical ability, TSIK appears to be a viable and competitive alternative in securing web based business interactions.

#### 3.2.3.1 Software analysis

Java *keytool*, Java's key and certificate management tool, is used to create the Java keystore, with appropriate key pairs, used by TSIK and JCE. The keytool generates key pairs for RSA where $N$ is user specified (512, 1024 or 2048), $d$ is the same length as $N$ and $e$ defaults to 216 + 1 (i.e. 17 bits long). As stated in Section 3.1.4, these values are adequate and it is currently recommended that the user selects the modulus to be at least 1024 bits.

TSIK 1.10 provides additional functionality, above that of the Java Cryptography Extensions (JCE), to construct valid XML messages after encryption/decryption or signing/verifying. These messages conform to the W3C XML Signature and Encryption specifications [97]. TSIK supports Triple DES (in cipher block chaining mode) for symmetric encryption, as defined by W3C [97]. Using a key length of at least 112 bits will currently provide sufficient security. Triple DES is however relatively slow compared to other more recent contenders such as AES [98]. Conversely, it has stood the test of time and so is potentially a more reliable solution.

Only SHA-1 is provided for message digest generation (digest length of 160 bits). SHA-1 has very recently been shown to be less secure than predicted and it is recommended that SHA-256 or better should be used [94]. RSAES-PKCS1-v1_5 algorithm, specified by W3C [97] and RFC2437 [99], is used as the RSA standard. As stated in Section 3.1.4, if backward compatibility is not an issue Optimal Asymmetric Encryption Padding (OAEP) should be used in preference to PKCS1-v1_5. However, PKCS1-v1_5 provides adequate security assuming the programmer is aware of certain issues. Also, RFC2437 [99] indicates that RSACRT is used.

JCE does not support the creation of valid XML messages but supports various symmetric key algorithms including AES, Triple DES and RC5. It also supports SHA-1, SHA-256, SHA-512 and MD5, amongst others, for message digest generation. It also specifies that the padding is applied according to RFC3447 [76]. RSA-CRT is also used.

#### 3.2.3.2 Performance analysis

The following section details a comparative evaluation of the performance of VeriSign's TSIK toolkit and the standard Java Cryptography Extensions (JCE) in order to identify whether TSIK is a viable tool to secure web services, for instance those used in performance critical online transactions.

### Environment

All experiments were run on a 3GHz Intel Pentium 4 with 1GB RAM, running Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2-b28) on top of Linux Fedora Core 2. We used The Legion of the Bouncy Castle [100] as the Java RSA provider for both JCE and TSIK, and used Apache Axis 1.2 to generate the appropriate WSDL interface for the web service, which was hosted on Tomcat 5.

Axis was used to both generate the appropriate SOAP messages, from the Java code and TSIK XML documents, to be sent to the web service, on the server side, and to generate the SOAP messages which are sent back from the web service to the client. We took performance measurements on the client as well as server side. Message transmission and conversion delays were not measured.

### Experiments

In comparing the performance of TSIK and JCE we first consider message confidentiality, namely encrypting the message using 3DES and encrypting the 3DES key using RSA. Secondly we consider message integrity and authentication by evaluating the cost of signing and verifying messages using SHA-1 and RSA. Lastly we study non-repudiation and evaluate its cost as a function of key length. The three experiments are detailed below.

*Experiment 1:*

In experiment 1 we analyse the performance of Triple DES, as a function of message size:

- Client side: Message plaintext encrypted using Triple DES with a key size of 168. Symmetric key encrypted using an RSA public key (modulus 1024)
- Server side: Encrypted symmetric key decrypted using RSA private key (bit length 1024) and cipher text then decrypted.

*Experiment 2:*

In experiment 2 we analyse the combined performance of SHA-1 and RSA algorithms, as a function of the message size:

- Client side: Message signed using SHA-1 and RSA private key (bit length 1024)
- Server side: Message verified using SHA- 1 and RSA public key

*Experiment 3:*

In experiment 3 we analyse how the modulus size affects the performance of RSA during signature creation and verification:

- Client side: Message signed (as in experiment 2) using RSA key sizes 512, 1024 and 2048.
- Server side: Message verified.

**Results**

The experiments above were conducted for TSIK as well as JCE. We repeated the first two experiments for messages with a range of plaintext sizes, namely 2, 4, 8, 16, ..., 512 and 1024 KB. Experiment 3 was done using a 2 KB plaintext size. The results are shown in the figures below. It should be noted that all points in Figures 3.10 and 3.12 exhibit 90% confidence intervals of 3 milliseconds or less and points in Figures 3.11 and 3.13 exhibit confidence intervals of 0.1 milliseconds or less.

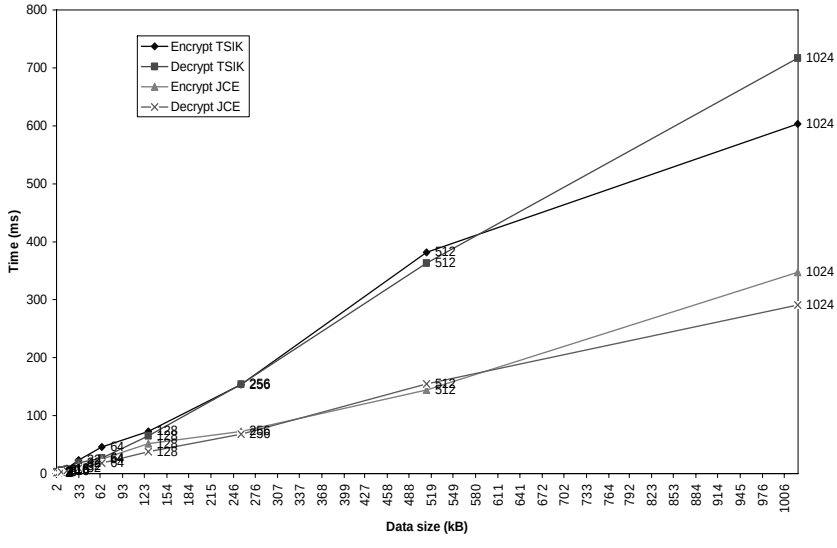


Figure 3.10: Triple DES encryption time

*Experiment 1:*

Figure 3.10 shows that JCE performs noticeably better for large file sizes. It also shows that Triple DES encryption takes longer than decryption in both cases (TSIK and JCE) except for very large messages where decryption takes longer when using TSIK. We have no precise explanation for this and can only suggest it has to do with the particulars of the implementation.

For RSA we see the opposite effect. Figure 3.11 indicates that RSA encryption takes less time than decryption. As explained in Section 3.1.4.1, that is caused by the size of the keys used in encryption and decryption. For encryption, the public key is used, which has a small public exponent of 17 bits. For decryption the large private key is used whose exponent is the same length as the modulus (i.e 1024 bits). When comparing TSIK with JCE, we see that the differences are minimal. Decryption varies by an average of about 1 millisecond between the implementations and encryption even less. We also observe that the cost of encrypting or decrypting the Triple DES key with RSA in Figure 3.11 is relatively small compared to the overall cost of encrypting/decrypting the message in Figure 3.10.

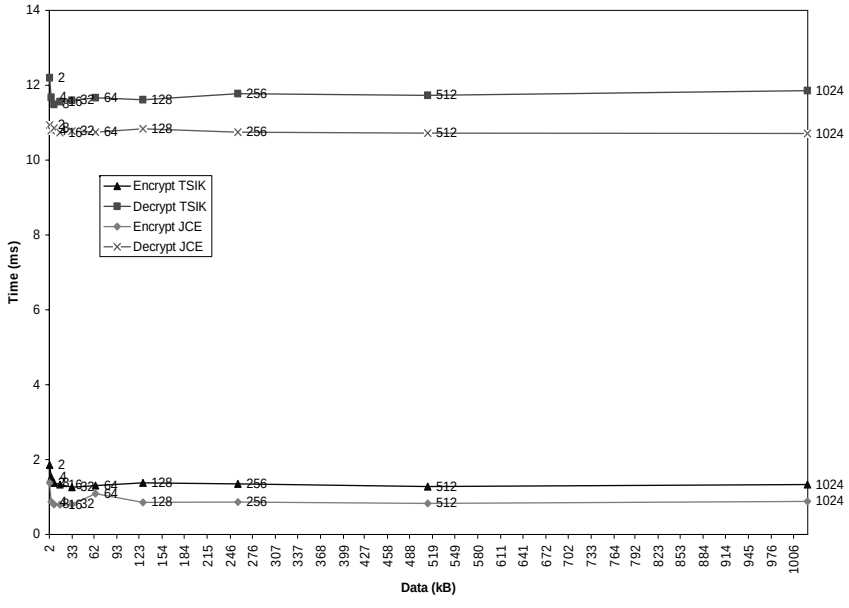


Figure 3.11: RSA-1024 encryption time of 168 bit Triple DES key

*Experiment 2:*

Figure 3.12 shows that as message size increases signing consistently takes more time than verification for both JCE and TSIK. This is once again expected as the messages are signed using the large 1024 bit RSA private key. Encrypting the message digest should take constant time for each file size and so the graph pattern should be wholly due to SHA-1 hashing. Whereas signing and verification time increase steadily for JCE, TSIK performs markedly worse for large file sizes. TSIK should also consider using the more secure SHA-256 for message verification in future releases as is discussed in Section 3.2.3.1. We also see a 20ms fixed difference in overhead for TSIK. This is likely due to the cost of converting the message into an XML readable format.

*Experiment 3:*

Figure 3.13 shows that doubling the RSA key size causes signing time to increase whilst having little effect on the verification time. This can be explained by the fact that doubling the key size effectively doubles the length of the private exponent (used in signing) whilst keeping the public exponent length constant.

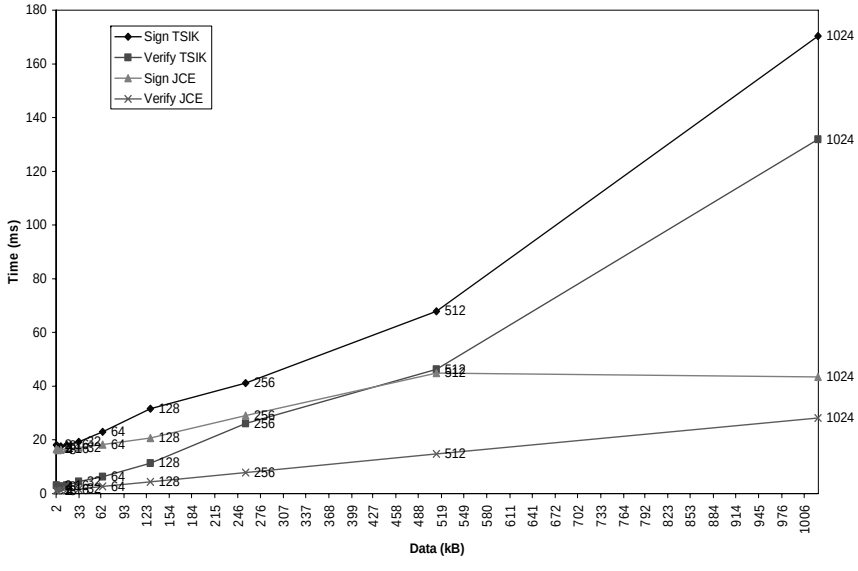


Figure 3.12: Message signing/verifying (using SHA-1 and RSA-1024)

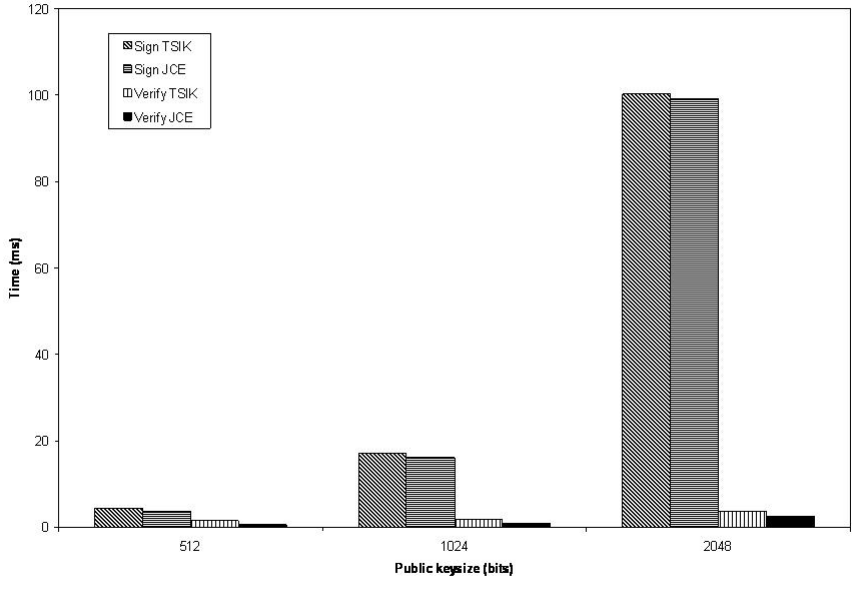


Figure 3.13: Message signing/verifying (2KB message size)

### 3.2.4 Summary

This chapter evaluated implementations of the cryptographic algorithms used to achieve message confidentiality, integrity, non-repudiation and authentication. It has consistently been found that the implementation has a significant impact on the expected performance of each algorithm.

We first considered two client side Java implementations, namely standard Java JCE libraries and Cryptix libraries, and found that the standard JCE libraries performed markedly worse. 3DES performed 25% worse than DES for Cryptix and only marginally worse for JCE. The expected performance degradation of 3DES is 50%, indicating that the implementation played a significant part in algorithm performance. IDEA was shown to be the fastest symmetric algorithm.

Secondly, two server side OpenSSL implementations were considered. AES-128 performed particularly well for the newer OpenSSL 0.9.8d version even outperforming IDEA. The MD protocol family showed performance increases from MD2 through to MD5 for OpenSSL 0.9.7.f and Cryptix, whereas the 0.9.8.d implementation peaked at MD4. This further emphasises the fact that the implementation choice influences the algorithm performance significantly.

Lastly we evaluated Verisign's TSIK toolkit for trusted Web Service interactions, comparing it to Java's standard JCE library. TSIK has comparable performance but suffers when signing/verifying larger files. It is likely that his can be remedied by a more careful implementation. Further improvements such as using the secure SHA-256 hashing or opting for faster symmetric encryption algorithms is also advisable [94].

Having taken a closer look at the performance and security of the algorithms used to secure client-server interactions we now turn to the security protocols which facilitate the interaction in the next chapter.

# Chapter 4

# Adaptive Security

Using the security algorithms analysed in the previous chapter as well as adaptive security principles discussed in Chapter 2 we now focus on the creation of an adaptive security solution. We first present a method to augment existing security services with adaptive features, outlining key steps and activities for such a process. The method itself is designed to achieve an adaptive security solution that adheres to the feedback loop mechanism put forward in IBM's Autonomic Computing model [63]. We furthermore put forward a generic design for an adaptive security service, detailing core interactions amongst key components. The methodology and design forms the basis for our Adaptive Security solution in Chapters 5 and 6.

## 4.1 Methodology

In this section we provide a methodology to achieve and adaptive security solution that conforms to IBM's Autonomic Computing vision. Extending a Security Service to facilitate security adaptation requires a number steps and considerations.

In addressing the scope of such a methodology we note that it in particular achieves adaptation for existing security systems. Much has been done in the pursuit of achieving runtime adaptation for new systems built from the ground up. Such solutions are typically platform specific (e.g. Dynamic TAO, Open ORB, etc.), limiting their scope to adaptive security systems on those platforms, or language specific (e.g. Open Java, R-Java, PCL, etc) limiting its use to security systems of those languages. The extension of existing security systems with adaptive features is addressed in our method-

ology.

In the context of existing security systems, our methodology addresses a certain subset of systems that can be considered for adaptation. Firstly it applies to systems where the cross-cutting concern is measurable property of the environment. We advocate a measurement approach where the relationship between security and the cross-cutting concern is studied offline through experimentation. The challenge lies in identifying a measurable property of the environment to represent the cross-cutting concern, assuming this is possible. For example, when measuring the environment for security threats before adapting firewall security policies one must first decide what should be measured to accurately represent a threat. Furthermore, this measurement must also be made at runtime, as is also the case for other runtime adaptive systems, for comparison with the offline data. It is therefore imperative that the performance cost of such monitoring should is not prohibitive to the adaptation process or the expected system functionality.

When considering the security adaptation itself we note that our methodology applies to systems where the security adaptation has an impact on the chosen cross-cutting concern. In other words, as detailed in Section 2.2.1, the impact of the adaptation on the security system serves as feedback to the Monitor component in the next feedback loop cycle. As such their relationship can be studied and automated security actions taken at runtime.

Lastly, a key consideration when applying the methodology to achieve runtime security adaptation, is how to successfully leverage control from the existing system. The pertinent part of the existing system is the decision process relating to what security mechanism to utilize and when to apply it. This is a key challenge and can range from trivially simple if the security system is built with or on top of adaptive technologies or prohibitively difficult for closed source legacy systems. A number of techniques on how this may be achieved is discussed in Chapter 2.1.2

### Methodology

1. Establish a *control point* in the existing system where a security adaptation can be induced. Effectively leveraging control is system specific and can be done in

a number of ways including providing wrappers (i.e. encapsulation) for existing system components or utilising any number of other techniques detailed in our background chapter, Section 2.1.3.

2. Identify a *cross-cutting concern* against which security will be traded off. Such a concern must be a measurable property of the environment or context in which the system operates. It should be measurable at runtime and monitored without adversely affecting functional or non-functional system properties.

3. Study the *interrelationship* between security and the cross-cutting concern in the existing system context. In particular identify measurable factors which most significantly influence their relationship. Assess such factors off-line trough experimentation building an accurate and relevant representation of the relationship, and therefore the system, based on such factors. Such data may at this stage be system or implementation specific. Trends based on the factors in the relationship can therefore be transposed to a model of the system which should then be further verified as to its accuracy.

4. Formulate a *trade-off goal* based on the desired security characteristics as well as cross-cutting concern considerations. Evaluating security is a complex and error prone process and is detailed in Section 1.2. Create an appropriate policy with steps based on the system representation or model to achieve the desired goal.

## 4.2 Design

Having specified the steps required to extend a security service with adaptive features we now detail a design of such a newly adaptive system to support the methodology process. We also provide a more detailed design of one particular design component, namely the *Adaptation Unit*, since this will be needed in the next chapter where we apply the principles in this chapter to the design and implementation of an adaptive Cryptographic Service. The security service in the design can represent any of the security types discussed in the previous chapter. In keeping with our methodology we present the following design:

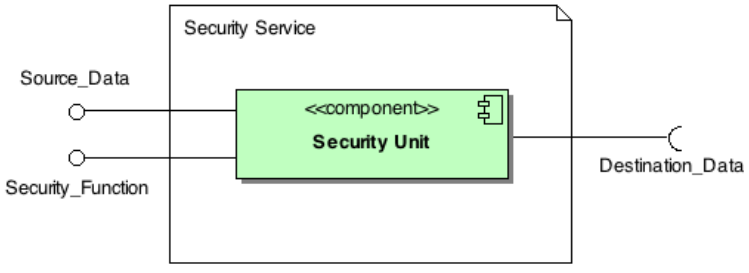


Figure 4.1: Security Service

Figure 4.1 depicts a standard security service. The *Security Unit* takes source data as input, applies the relevant security function to the data and returns the transformed data. The data transformation may be symmetric cryptography, asymmetric cryptography or hashing.

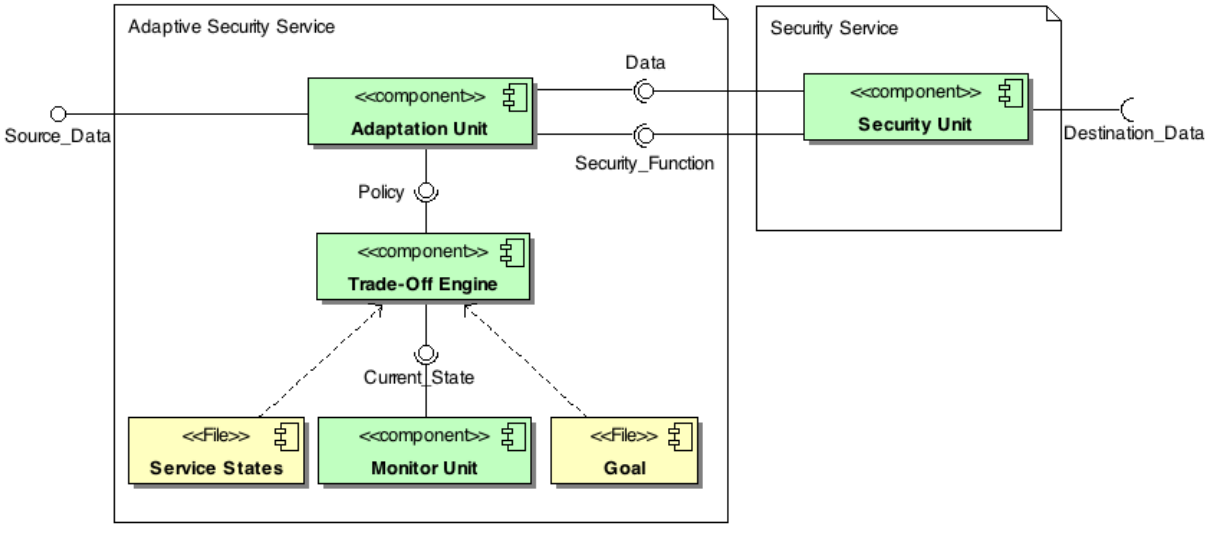


Figure 4.2: Adaptive Security Service

Extending such a service with adaptive features requires a number of key components and interactions. Figure 4.2 depicts the components of an adaptive security service which augments a standard security service with adaptive features. Furthermore, the design is a realisation of the adaptive feedback loop in Chapter 2.

The *Adaptation Unit* serves as a key component in the design. It is akin to the Executor service in the feedback loop and is responsible for enforcing the security adaptation. Firstly it is required to intercept the *Source_Data* in transit to the *Se-*

*curity Unit*, forming a control point where security can be adapted. Furthermore, it maps the given *Policy* rules to the *Source_Data* indicating to the *Security Unit* which *Security_function* to apply to which sets of *Data*.

The *Monitor Unit* enables the Adaptive Security Service to observe and report on a cross-cutting concern in the service environment. It corresponds to the Monitor component in the feedback loop. Through runtime monitoring of the system context it enables the *Trade-Off Engine* to make informed decisions based on the *Current_State* of the system.

The *Trade-Off Engine* component serves as the decision point for the next two components of the feedback loop. Firstly it represents the Analyse component; The *Service States* file contains pre-computed data or a system model which represents the interrelationship between security and the cross-cutting environmental concern. Taking the *Current_State* into account the *Trade-Off Engine* is thus able to determine the future system state, i.e. the change in the relationship, if security is adapted. The last component is the Plan component and determines if and how security should be adapted based on the adaptation *Goal*. With reference to the *Service States* the *Trade-Off Engine* generates a *Policy* which satisfies the adaptation *Goal*.

### Adaption Unit Design

In this section we explore the *Adaptation Unit* further in support of our adaptive Cryptographic Service design in Section 5.2.

The *Adaptation Unit* is instrumental in the adaptation process in two key areas. Firstly it is tasked to intercept data destined for the *Security Service* thus leveraging control over which security transformations may be applied to this data in future. Additionally, with the aid of a given set of policy specifications, it instructs the *Security Service* on which security functions are appropriate for which sets of data. Figure 4.3 depicts the primary interfaces to such a component (Also see Figure 4.2 for the Adaptation Unit in context of the adaptive security service).

In aid of creating an appropriate design for a Cryptographic Service's Execute component in the next chapter (Section 5.2) we now further define the structure of the generic *Adaptation Unit*. Such a design can be seen in Figure 4.4.

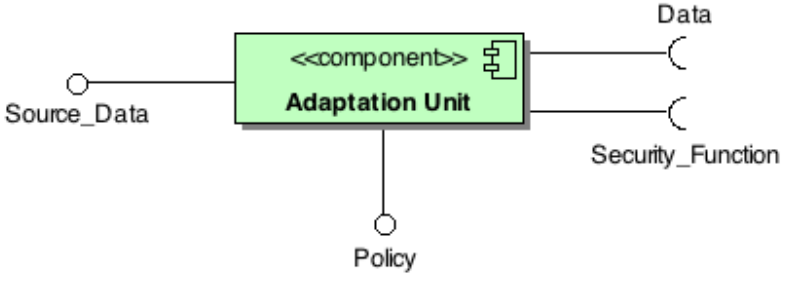


Figure 4.3: Adaptation Unit

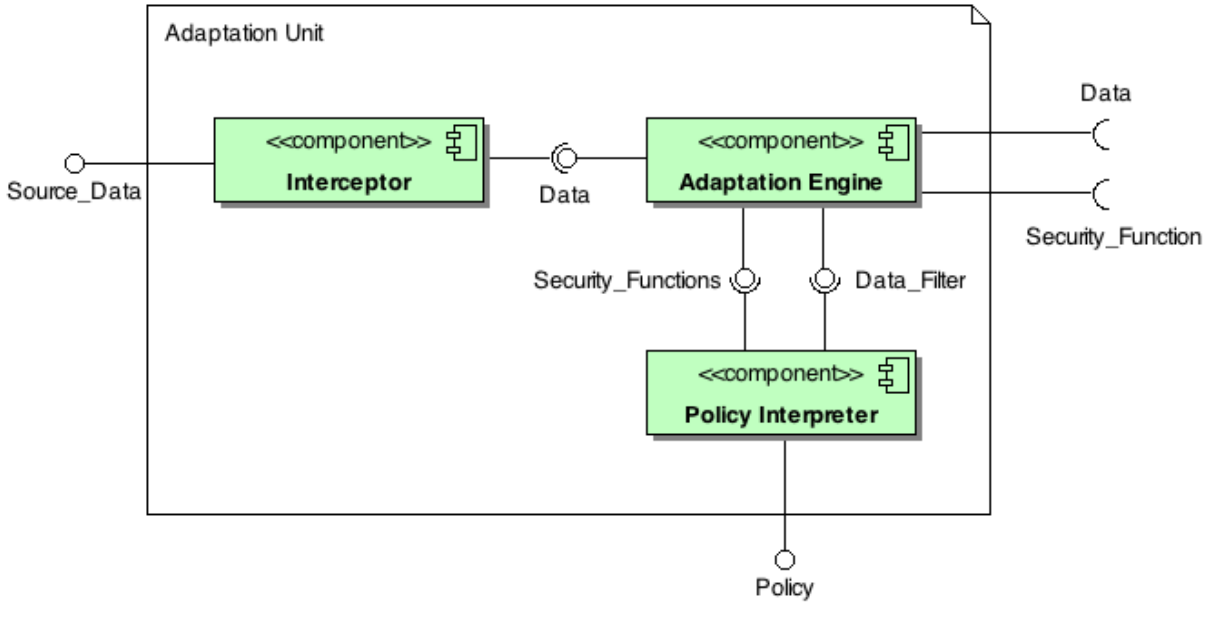


Figure 4.4: Adaptation Unit Components

The leftmost component called the *Interceptor* diverts data destined for the *Security Unit* to the *Adaptation Engine*. Data may be parsed and presented in a format specified by the *Adaptation Engine* interface.

The *Policy Interpreter* parses the incoming policy updates, generating appropriately formatted *Security_Function* & *Data_Filter* pairs as input to the *Adaptation Engine*. Such parsing may include, but is not limited to, authenticating the sender, policy syntax checking and mapping policy rules to internal security algorithm & data filter representations.

Lastly the *Adaptation Engine* applies the filter conditions specified in the *Data_Filter* to the incoming data from the *Interceptor*. Once filtered, the *Data* is sent to the *Security Unit* specifying the appropriate *Security_Function* to apply to the particular subset of data.

## 4.3 Summary

In this Chapter we introduced a methodology to successfully create an Adaptive Security Service. To that end we also presented a design to facilitate the creation of such a service which adheres to the adaptive systems feedback loop design.

In the next chapter we explore the design and creation of an *Adaptation Unit* for a particular service environment (i.e. Web server with SSL/TLS cryptographic security). In the chapter thereafter we delve deeper into the other stages of the feedback loop, exploring the interrelationship between security and performance through experimentation and finally demonstrating the effectiveness of such a trade-off at runtime.

# Chapter 5

# Adaptive Security for SSL

In this chapter the first step of the methodology in Section 4.1 is applied to create a control point through which a security adaptation can be applied to an existing Cryptographic security service. In particular, we focus on the realising an *Adaptation Unit* for the SSL cryptographic protocol in a web server context by applying our design in Section 4.2 which supports our methodology. The solution augments cryptographic security measures between clients and a web server with adaptive features to respond to various runtime security influences in a timely and effective manner.

We first introduce the SSL protocol, which utilises many of the cryptographic algorithms analysed in Chapter 3 to secure communications between a client and server. We then realise our Adaptive SSL (ASSL) solution through a design and implementation which reflects the *Adaptation Unit* design in Section 4.2. Finally we evaluate its performance to ascertain its viability as an effective adaptive SSL solution.

## 5.1 SSL

Secure Socket Layer (SSL), also referred to as Transport Layer Security (TLS), is a protocol used to secure communications between an application or web server and a client over the Internet. The protocol is standardised through the Transport Layer Security [69] effort and involves establishing a secure transport layer connection between a client and server through a handshake mechanism. Security is provided in the form of authentication, confidentiality, integrity and non-repudiation of messages.

During the handshake, algorithms are selected for the aforementioned security properties, based on those available to both the client and the server. This handshake process is commonly known as *SSL negotiation* and the resulting secured connection is called a *session*. Established SSL sessions can also be *renegotiated* at the discretion of the client or server.

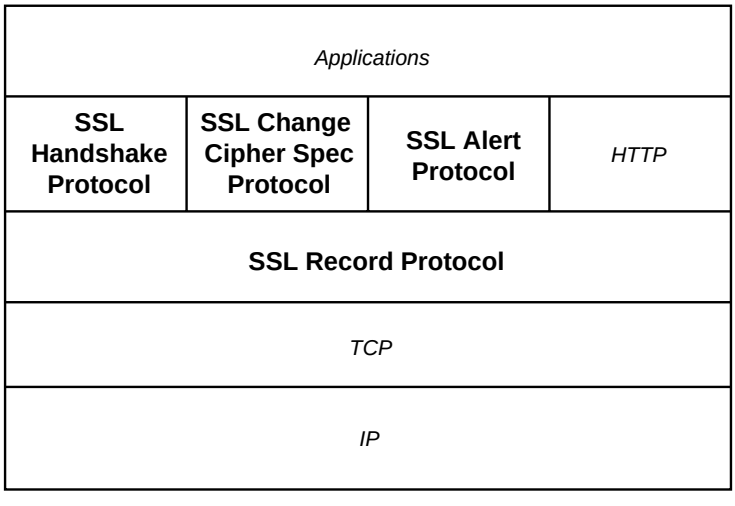


Figure 5.1: SSL Protocol Stack [3]

### 5.1.1 Protocol

The SSL protocol is comprised of a number of protocols and protocol layers which facilitate the establishment of a SSL *session* between a client and a server. The server can be any application, such as a web server. In this section we look at how the different protocols compliment each other to provide this secure SSL service [3, 69].

Figure 5.1 shows the SSL protocols in relation to the other internet protocols. The SSL Record Protocol provides security in the form of confidentiality and message integrity to the layers above. Of particular interest is the Hypertext Transfer Protocol (HTTP) which operates on top of the SSL Record Layer to provide secure Web based interactions. Three further protocols are specified as part of SSL. The SSL Handshake, SSL Change Cipher Spec and SSL Alert protocols are defined at a higher level to support management of the SSL secured sessions. We discuss these protocols in more

detail below.

#### 5.1.1.1 SSL Record Protocol

The SSL Record Protocol provides a secure client-server connection for higher layer protocols. It ensures data confidentiality through symmetric encryption and message integrity through hashing. Messages from the layer above are first fragmented into blocks, then optionally compressed, a hash value added, then encrypted (adding padding as necessary) and finally sent. Upon receiving the data the process is reversed. Data is decrypted, verified, optionally decompressed, reassembled and passed to the higher layers.

#### 5.1.1.2 SSL handshaking protocols

The following three subprotocols facilitate negotiation of security parameters used by the SSL Record Protocol, instantiating negotiated security parameters, optional mutual authentication and reporting on error conditions. They work together to establish a secure session between the client and server.

### Handshake Protocol

The handshake protocol plays a key part in the SSL protocol and is responsible for negotiating the security parameters used by the SSL Record Protocol to secure the session. During parameter establishment the two parties may also authenticate each other and the session may also be renegotiated. A breakdown of the required client-server interactions can be seen in Figure 5.2. The protocol can be separated into four logical phases as indicated by the dotted lines. In phase one the client and server determine the encryption, authentication and compression algorithms to be used. In phases two and three the server and client respectively may be authenticated if required by the other party. Lastly the negotiated algorithms are activated and used to finish the negotiation process in phase four. Protocol details below are provided for completeness and are not strictly necessary to understand our research contribution. SSL terms are indicated in italics and optional messages are shaded in Figure 5.2.

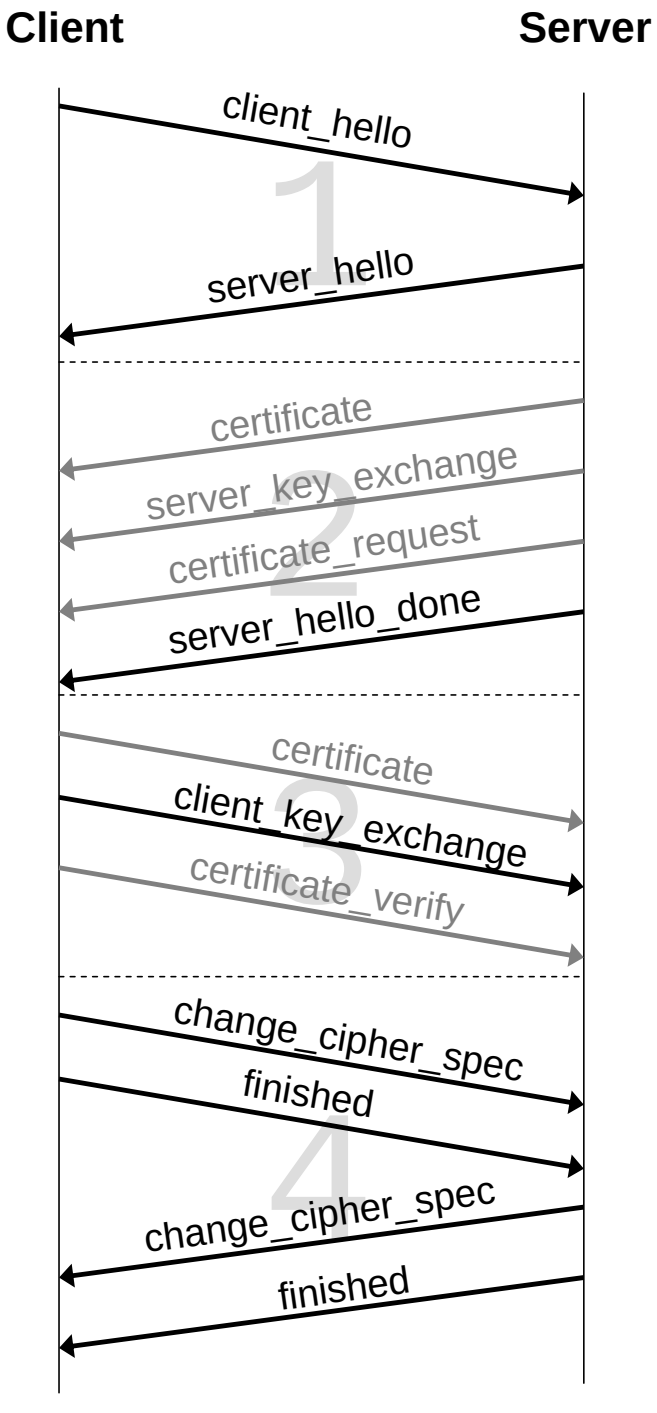




Figure 5.2: SSL Negotiation [3]

The client initiates the first phase of the negotiation with a *client_hello* message. The *client_hallo* message contains a *CipherSuite* list, in preferential order, of all supported algorithms for the main cryptographic techniques. It also contains a random structure, lists the supported compression methods and provides the current SSL version number. Lastly it provides a *SessionID* to identify the session. Upon receiving the message the server selects a *CipherSuite* and compression method from the respective lists and replies to the client with a *server_hello* message. In addition to the preferred *CipherSuite* and compression algorithm selected from the lists it also contains a SSL version number, a newly generated random structure and a *SessionID*. If the returned *SessionID* is the same as that received from the client it implies that the server found the *SessionID* in the server session cache and the old session with that ID will be resumed (Protocol continues from phase four if this is the case). Otherwise the new *SessionID* returned becomes the current *SessionID*.

In phase two, if requested by the client, the server will respond with a message to allow the client to authenticate the server. Optional messages are shaded in Figure 5.2. The *certificate* message contains the server certificate (in a format specified by the *CipherSuite*) and a list of any additional required certificates. An additional *server_key_exchange* massage may also be sent depending on the authentication method (Further details are extraneous and not covered here). Once the server is authenticated it may request, through the *certificate_request* message, the client authenticate itself also. Finally the server sends a compulsory *server_hello_done* message to end the server hello message sequence.

If requested by the server, the client must send its own *certificate* message. The client must thereafter send a *client_key_exchange* message containing relevant key information which depends on the previously selected asymmetric protocol. If a certificate was sent the client sends a *certificate_verify* message which provides explicit verification that the client owns the certificate. This is done through sending a signed (using the client private key and hash function) version of the previous messages to the server.

Phase four is the final phase of the handshake protocol and completes the client-server session setup. For the first message the Change Cipher Spec Protocol (see

below) is invoked to activate the negotiated security parameters for all subsequent messages and notify the server thereof through a *change_cipher_spec* message. Lastly the client sends a *finished* message which is the first message exchanged with the new negotiated SSL session. The server similarly invokes the Change Cipher Spec Protocol and sends its own *finished* message.

From this point on the application layer can start sending messages using the newly negotiated SSL session.

Once all four phases have been completed either the server or client can ask for the session to be renegotiated by sending a *hello_request* or *client_hello* message respectively. This will initiate the SSL negotiation protocol but since the client and server have already authenticated each other and exchanged the necessary keys, the protocol completes phase one and continues from phase four where the newly agreed on *CipherSuite* and compression algorithm can be activated. The role this feature plays in our Adaptive SSL solution is discussed later in this chapter.

### Change Cipher Spec Protocol

The Change Cipher Spec Protocol is responsible for changing the pending Cipher Spec state (the parameters negotiated during the handshake) to the active Cipher Spec state (the parameters used to secure the session). In Figure 5.2 the client sends a *change_cipher_spec* message using the pending Cipher Spec and immediately notifies the client SSL Record Layer to make the pending Cipher Spec state the active Cipher Spec state. Once both client and server have invoked the Change Cipher Spec Protocol they will be able to send and receive messages using the newly active Cipher Spec state.

### Alert Protocol

The alert protocol is used to communicate a variety of fatal or warning messages to the other party. Messages include closure alert messages such as the *close_notify* message to signify that the sender will not send anymore messages and close the connection. There are also a variety of error alerts that are used during and after

the negotiation phase. Such message include *handshake_failure*, *certificate_expired*, *decryption_failed*, *internal_error*, etc. See RFC4346 [69] for full listing.

## 5.2 Adaptive SSL

The *Adaptation Unit* is a key component in providing Adaptive Security for our chosen Cryptographical Service, namely SSL. In this section we explore the design, implementation and performance evaluation of such a component which we henceforth refer to as Adaptive SSL (ASSL).

We first detail our ASSL design, followed by the implementation and finally an overhead evaluation to ascertain its suitability for runtime security adaptation.

### 5.2.1 Design

This section explores the design of an ASSL Adaptation Unit in a web server context. We chose to design ASSL for the Apache web server environment. Apache is a popular web server used in industry today. As such our ASSL solution spans the gap from research to industry making it a practical tool for runtime security adaptation. Apache is also an open source web server platform and so allows greater flexibility in how we can implement our design.

Our ASSL design facilitates runtime security changes to SSL secured sessions in response to cross-cutting environmental concerns. We first evaluate current methods of changing security and present a better alternative as a further motivation for our design. Next we detail the design showing how ASSL can change runtime security based on a variety of environmental concerns. Finally the design is show to be consistent with the generic Adaptation Unit design in Section 4.2.

#### Design motivation

Web server security is typically configured before the server is started. It is however possible to change the security at runtime using distributed configuration files. These configuration files are distributed throughout the server file system and provide the configuration directives for the directories and subdirectories they reside in. The

configuration files can contain directives for any server aspect including security and can be changed by anyone who has write access to the file. These files are therefore ideally suited to scenarios where users need to configure their part of the server but do not have access to the main server configuration file. A typical example is where Internet Service Providers (ISP) host multiple user sites and want to give users permission to configure their own sites.

Clearly this is a far cry from our envisioned Adaptive SSL solution where security adaptation is automated and can react in response to a variety of environmental concerns such as threat levels or performance considerations. Distributed configuration files are however also not suitable as a basic building block for our ASSL solution. Firstly, these files allow configuration directives other than those related to security and so permitting their use might result in unexpected changes to the server configuration by web site owners.

Secondly, allowing their use incurs an additional performance cost for the server. For example, if a client requests the file index.html in directory /www/htdocs/example the presence of the following configuration files have to be checked:

- /configuration
- /www/configuration
- /www/htdocs/configuration
- /www/htdocs/example/configuration

Additionally if any are found their configuration directives are read and merged, according to a rule set, into a single configuration file. This process is repeated for every client request regardless of whether the configuration files have changed.

The directory accessed determines the security directives that are applied. Configurations may additionally specify a filter on file types to which the security directives will apply. Adapting security only on the directory and file type accessed is clearly quite restrictive and we may wish to adapt security based on connection- or client information.

### Requirements

In a design of an Adaptive SSL solution we would therefore endeavor to provide a certain set of key features in addition to meeting the Adaptation Unit design requirements detailed in Section 4.2. Firstly we need to sperate the concern of security from that of server configuration. Decoupling the security rules in this way from the main server configuration allows us to build a more powerful and flexible adaptive security model since the security rules can be determined, deployed and changed without restriction, independently and parallel to the web server and its components.

Secondly, the design must facilitate the identification of clients requiring security adaptation using an extensive set of filter conditions. This will allow for more expressive security rules and facilitate a closer match between the environmental conditions being monitored and the actual clients that need a security change. For example if security threats are monitored, clients requiring additional security could be identified by their IP address. Identifying clients based on a subset of connection- or client information rather than the limited file location and type would achieve this.

Finally Adaptive SSL's design must accommodate security adaptation based on a wide range of cross-cutting concerns. Such concerns should not be stipulated by ASSL and so place no restrictions on the intended use of ASSL. Concerns may be monitored by specialised $3^{rd}$ parties thus removing the logic that guides the adaptation from the main server. "As a separate entity, the effectiveness of the adaptation logic is more analysable and the mechanism more modifiable and extendable"[27]. Specialised $3^{rd}$ parties could include firewalls which adapt security based on current threat levels. For example, clients that use typically insecure wireless internet connections at airports can be provided with a higher level of security if the $3^{rd}$ party detects that the client is using the server from the airport or if the airport firewall detects a security threat. Others could include system performance monitors which maximise the security based on current resource availability, system administrators who need to respond to a threat quickly or data monitors which alter the connection security based on the security requirements of the data transferred. We further explore one of these, namely adapting security based on current system performance, in the next chapter. ASSL adaptation should also be controlled through a standard interface allowing $3^{rd}$ parties

to design their components in a "Commercial-off-the-shelf" (COTS) fashion for later integration with the system.

Designing ASSL in this way adheres to many concepts discussed in Chapter 2 and allows it to form a vital part of the self-adaptation feedback loop by providing the execution component of such a loop.

### ASSL design

Applying key ideas and concepts in the previous section, we present our Adaptive SSL design in the context of a web server environment.

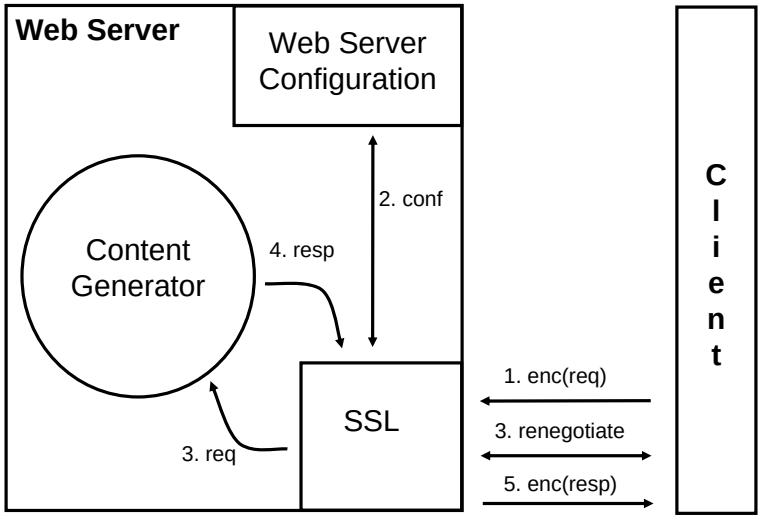


Figure 5.3: SSL (enc = encrypted; req = request; resp = response; conf = configuration)

Figure 5.3 depicts a standard web server request-response processing cycle during a typical SSL secured session. The numbers in the figures indicate the event order and the labels the interaction type. Events with the same number indicate a decision point and only one of the events take place. Figure 5.3 shows the client sending an encrypted request to the server in step 1. The request is initially passed to SSL which then queries the web server's configuration file (where the security rules are stored) in step 2. In step 3 SSL either decides to adapt the security by renegotiating the SSL session or passes the request to the content generator. Depending on the request the content generator may either reply with a created file or provide a static file as

a response in step 4. Lastly, the file is encrypted by SSL and passed securely to the client.

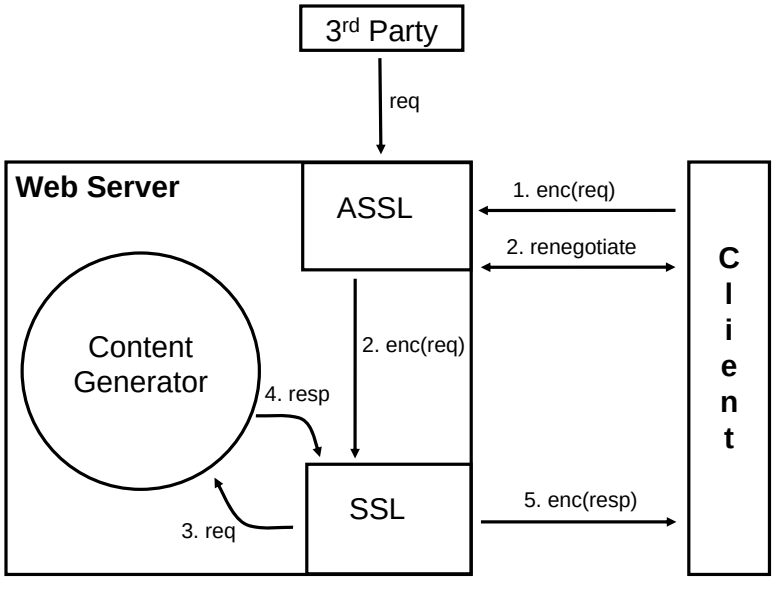


Figure 5.4: Adaptive SSL

Figure 5.4 shows how ASSL is incorporated into the web server environment. It effectively takes over security adaptation, namely the SSL negotiation and renegotiation logic, by intercepting requests and evaluating them against the negotiation rules specified by $3^{rd}$ parties. ASSL intercepts the client request in step 1 and either renegotiates security for the SSL session or passes the request to SSL in step 2. Input from $3^{rd}$ parties are not numbered in the figure as they are allowed to send requests to change the negotiation rules at any time. This has far reaching implications as it allows security to be tightly coupled with runtime environmental monitoring and so allows the security to change as the environment or security requirements change.

Figure 5.5 is a magnified version of the ASSL module in Figure 5.4. It shows how ASSL adapts the session security, during the request-response cycle in Figure 5.4, based on renegotiation rules specified in the Request Filter. As can be seen in step 2, the request filter allows security updates from $3^{rd}$ parties and client queries to run in parallel. $3^{rd}$ parties are therefore able to identify clients, or sets of clients, at runtime in a flexible and effective manner. Request Filter security conditions could include client security level, type of client, client location, client name, etc. See next chapter

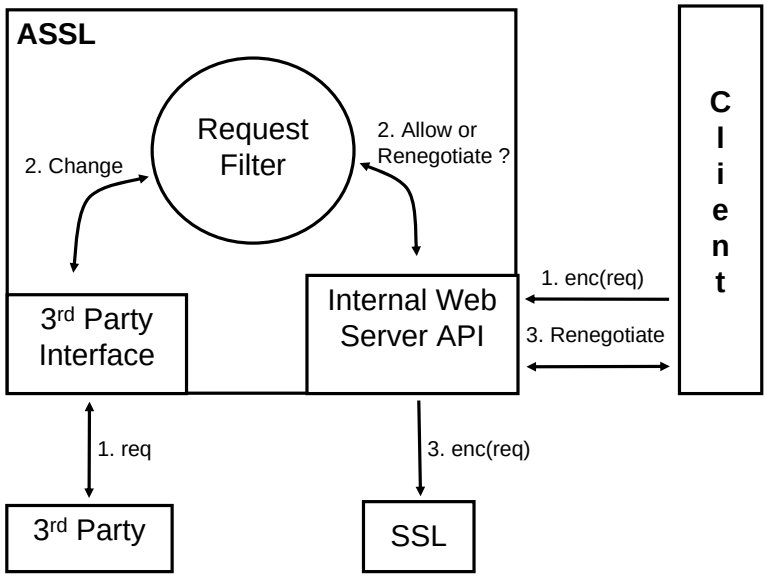


Figure 5.5: ASSL session security

for details of the chosen Request Filter implementation.

### Design evaluation

This section showed how ASSL can provide a runtime adaptive security solution for SSL in a web server context. We now evaluate the domain specific design above with respect to its adherence to the generic *Adaptation Unit* design, as detailed in Section 4.2, as well as showing that our design additionally meets the requirements stipulated earlier in this chapter.

Firstly it conforms to the *Adaptation Unit* design. As seen in Step 1 Figure 5.4 intercepting client requests destined for the SSL component fulfills its role as *Interceptor*. ASSL furthermore supports incoming policy updates through a $3^{rd}$ party interface as seen in Figure 5.5. This functionality is akin to that of the *Policy Interpreter*, accepting and parsing the incoming policy updates. Lastly, ASSL applies the policy updates stored in the request filter to data associated with a particular client. Such filtering and the associated application of a particular cryptographic algorithm to the data adheres to the functional requirements of the *Adaptation Engine* component.

The ASSL design furthermore adheres to a number of additional requirements de-

tailed in this chapter. Firstly it separates the concern of security from that of server configuration by supporting SSL security configuration through the ASSL request filter. Security configuration have thus moved from a webserver centric SSL configuration to one that is configured and maintained by the ASSL component. ASSL is also required to support the identification of clients through a set of extensive filter conditions. It does this through the Request Filter shown in Figure 5.5. Request Filter particulars are detailed for our chosen implementation in the next section. Lastly the ASSL design accommodates security adaptation based on a wide range of cross-cutting concerns through the HTTP interface which is exposed to COTS based $3^{rd}$ parties. Any cross-cutting concern can be monitored and reacted on (i.e. updating ASSL security) by these specialised $3^{rd}$ parties.

### 5.2.2 Implementation

In this section we present and evaluate key features of the ASSL Apache based implementation.

The Apache [96] web server is built on a modular design where nearly all of its functionality is provided through modules. Modules may register an interest to manipulate a client's request at various points during the request-response processing cycle. Apache provides various hooks to facilitate this. Apache is configured using *directives* which are read from a file at start-up (*httpd.conf*) or at runtime through the distributed *.htaccess* configuration files which are stored in the particular directories accessed during a client request. Modules may also have their own directives which can be included in these files. Htaccess files are however not ideally suited to automated runtime configuration as discussed in Section 5.2.1.

SSL is one such Apache module which integrates the OpenSSL toolkit [86] into Apache. SSL session security is configured through the *SSLCipherSuite* directive. This directive specifies a subset of security algorithms that can be used to establish a secure session with the client. A directive to include all algorithms in order of strength would look as follows: `ALL:+HIGH:+MEDIUM:+LOW:+EXP:+NULL`. This is a custom syntax particular to OpenSSL. See Appendix B for a full description.

Adaptive SSL is also implemented as an Apache module and can be installed on

existing Apache installations as it requires no additional Apache or SSL source code changes. The significant parts are detailed below:

**Request Filter:** The request filter seen in Figure 5.5 is a sequence of [condition, SSLCipherSuite] pairs on which the security of a session with the client will be renegotiated if one of the conditions matches the client request and the current session security is not a subset of the newly selected SSLCipherSuite. Each condition in the list is checked in turn and the first that matches is selected, much in the same way as firewall filters. Conditions are formulated using the powerful *SSLRequire* directive (see Appendix A). Each condition is an arbitrarily complex boolean expression which can make use of standard CGI, Apache and SSL related variables. For example; A condition for the cipher suite `DES-CBC-SHA` (DES encryption in CBC mode using the SHA hashing algorithm) could be stated as follows [101]:

```
 (%{SSL_CIPHER} !~ m/^(EXP|NULL)/
and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd."
and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"}
and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5
and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 18)
or %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+$/
```

The above condition states that SSL requests with ciphers which are not of type `EXPORT` or `NULL`, from the organisation ‘‘`Snake Oil, Ltd.`’’ who are also from the organisational units `Staff`, `CA` or `Dev` and who makes requests Monday to Friday between 8 a.m. and 6 p.m. or any request which comes from the address 192.76.162 will need to be renegotiated according the *SSLCipherSuite* in the [condition, SSLCipherSuite] pair. In this example it is `DES-CBC-SHA`. See Appendix A for the full list of variables that can be used in a *SSLRequire* directive.

The *SSLRequire* directives are expressed using an extensive subset of contextual information at the right level of abstraction. It uses only contextual information relating to the end-to-end client-server connection (which is the primary concern of SSL), nothing higher (application context) and nothing lower (data context). This

separation of concerns allows the module to be used in a variety of application and data contexts without any unnecessary or artificial restrictions at those levels of abstraction.

$3^{rd}$ **Party interface** ASSL also registers a handler, which manages all $3^{rd}$ party requests to Apache to insert condition pairs in the Request Filter. This provides a platform independent and secure means (allowing $3^{rd}$ party verification) by which the security can be changed as all $3^{rd}$ party requests are made over HTTP. A request to change the security is a delimited URL specifying the location in the Request Filter to insert the condition pair, the SSLRequire condition and lastly the SSLCipherSuite to apply if the SSLRequire condition matches the request. In the example below the request is made to the server at *domain.com* and *adapt-ssl* indicates that the request must be processed by the ASSL handler. This is followed by Request Filter location (i.e. 1) where the condition pair will be inserted. The [*SSLRequire*, *SSLCipherSuite*] condition pair is an example of reducing server security to no more than 56 bit encryption. It states that all clients using key sizes greater than 56 bits should renegotiate their security to use DES encryption (i.e. 56 bit encryption) and SHA hashing.

```
http://domain.com/adapt-ssl?"1?%{SSL_CIPHER_USEKEYSIZE} > 56?DES-CBC-SHA"
```

The handler provides a thread safe means to alter the Request Filter and so facilitate runtime security adaptation based on runtime configurable conditions. It also alleviates the burden on the web server to manage session security and allows specialised programs and individuals who can more effectively monitor environmental factors, such as potential threats and system load, to alter the security. Such parties could include firewalls, system performance monitors, network monitors and system administrators. Resolving conflicts of concern between the various parties are not within the scope of this thesis.

**Hooks:** ASSL registers a number of hooks with Apache in such a way that they are interleaved with existing SSL hooks, in essence taking control of the SSL renegotiation functionality without altering the existing SSL implementation.

**Configuration.** SSL's *SSLCipherSuite* directive is stored in the event that it needs to be applied, when appropriate, if the Request Filter (see above) is empty or none of the conditions in the Request Filter match the client request. Thus logically reverting back to the state where SSL has control over session renegotiations.

#### 5.2.2.1 Discussion

This section evaluates our Adaptive SSL implementation choices. We identify key strengths and limitations of our ASSL module.

Firstly, ASSL can be deployed on any SSL enabled Apache server requiring no changes to the server code. Only a server restart is required rather than a reinstall, as is required by some other modules, and so minimising barriers for ASSL deployment.

The ASSL module makes no changes to the existing Apache SSL implementation (i.e. mod_ssl [102]) and so need not be re-released when Apache SSL changes. Neither does it require the existing SSL implementation to be disabled or removed. ASSL leverages parts of the SSL implementation without affecting its operation and so ASSL additionally benefits from SSL updates such as bug fixes and features.

ASSL is also completely transparent to the client, requiring no specialised software on the client side. This significantly lowers the bar for ASSL adoption. Clients can, however, have moderated control of the SSL connection through an interface provided to them by the specialised $3rd$ parties who control the Web Server security (see paragraph below). In either case, clients maintain the right not to send data on connections below a certain security threshold by either examining the negotiated connection or simply limiting the algorithms that they claim to support during the negotiation process.

No restrictions are placed on the type of specialised $3rd$ party that can control the session security, though $3rd$ parties can be authenticated. ASSL does not resolve conflicts of interest between $3rd$ parties. Advocating $3rd$ party SSL control separates the concerns of SSL security from that of server configuration and so allows truly concurrent development cycles. The use of $3rd$ parties also lends itself to creative and potentially unexpected ASSL usage contexts.

Using SSL's *SSLRequire* directive provides adaptation decision rules based on the

client-server connection. In addition to allowing extensive adaptation decision rules through arbitrarily complex boolean expressions it also provides this at an appropriate level of abstraction, clearly separating the potential adaptation concerns. It allows connection adaptation based on connection parameters. Nothing higher (see Figure 5.1), allowing $3rd$ parties to consider application specific concerns such as performance issues or client preferences. And nothing lower, allowing $3rd$ parties to manage data specific considerations such as securing data based on the value of the information. Due to a bug in the implementation, *SSLRequire* does however require the server to create client processes using a non-threaded, pre-forking Multi-Processing Module. Although threading does enable greater scalability the prefork module is the Apache server default for Linux systems due to its stability and backward compatibility, amongst others [103].

$3rd$ parties interact with the server through the standard HTTP protocol. This gives $3rd$ parties the freedom to write their applications in any language on any platform. Applications also need not be hosted on the server machine thus allowing remote server security administration. ASSL supports $3rd$ party authentication but does not support encrypted security adaptation requests as the adaptation request is sent as part of the request URL. This is only a problem if the $3rd$ party is remote and ASSL can easily be extended to support this by including the request in the message body.

### 5.2.3 Overhead Evaluation

The following section details a performance overhead evaluation of our Adaptive SSL module to ascertain its suitability for runtime SSL session management. The chief overhead concern is that of evaluating client requests against the Request Filter as this process is executed for every client request. Renegotiating the session security when a match is found is handled in exactly the same way by both Apache's SSL and ASSL and as such is not considered here. In each experiment we compare Apache's SSL implementation to a 'best' and 'worst' case performance scenario for Adaptive SSL. In the best case the Request Filter list is empty and in the worst case the Request Filter list is full, containing twenty of the following complex boolean expressions.

```
%{SSL_CIPHER} =~ m/^(EXP|NULL)/
and %{SSL_CIPHER} =~ m/AES256/
and %{SSL_CIPHER_USEKEYSIZE} < 52
and %{REMOTE_ADDR} =~ m/^192.76.162.[0-9]+$/
and %{REQUEST_FILENAME} =~ m/secure/
and %{REMOTE_URI} =~ m/.mov/
and %{HTTP_USER_AGENT} =~ m/^Mozilla/
```

When evaluating a client request, each expression evaluates to false and so every expression in the Request Filter is evaluated for each client request. ASSL+ in Table 5.1 and in related figures in this chapter represents ASSL with a full Request Filter (the worst case scenario).

We first consider the security overhead for a single client request in experiment 1 to gain insight in the absolute time spent in the ASSL module. Experiment 2 evaluates the effect of this overhead on the server by stressing the server to determine the maximum number of requests it can process at any one time.

**Experimental environment.** All experiments were conducted on a 2.80GHz Intel Pentium 4 with 2GB RAM, running Apache 2.2.3 on Linux Fedora Core 5 (Kernel 2.6.17.11). SSL negotiations were performed with a 1024 bit RSA key and the RSA-DES-SHA1 cipher suite was utilised. OpenSSL 0.9.8d toolkit was used by

both Apache's SSL module and our module. Client workload was generated using HTTPerf [104] and Autobench [105] was used to simulate a request flow from multiple clients located on a number of machines. All requests were for a 44 byte index.html file.

#### 5.2.3.1 Experiment 1

Table 5.1 shows the average time the server takes to process various stages of the client request before content for the response can be generated. Stages include the server processing time consumed during negotiation, renegotiation and request decryption. "Wait for client" indicates server idle time during the negotiation between client and server. An * indicates that the values are the same as for SSL.

| Stage | Protocol | | |
|---|---|---|---|
| | *SSL* | *ASSL* | *ASSL+* |
| Negotiation | 115 | * | * |
| Renegotiation | 2 | 61 | 423 |
| Wait for client | 71768 | * | * |
| Decryption | 354 | * | * |

Table 5.1: Average processing times, in microseconds.

The table shows that in the best case scenario ASSL introduces about 60 microseconds overhead in the renegotiation phase compared to SSL. This increases to up to close to half a millisecond (423 microseconds) when the Request Filter demands more processing. To put this into perspective, the table shows that these amounts are easily outweighed by the time SSL spends in waiting for the client for instance. The overhead of ASSL in absolute numbers thus seems very minor. Note also that although SSL's use of distributed *.htaccess* configuration files for runtime security changes does not show up in SSL renegotiation time, it does increase Apache's request processing time depending on the size of the *.htaccess* file. This overhead is avoided by ASSL.

#### 5.2.3.2 Experiment 2

Experiment 2 evaluates the security overhead when the server is under load. We first note that the maximum number of requests the server can process at any one time depends greatly on client usage patterns. We therefore conduct two experiments, one

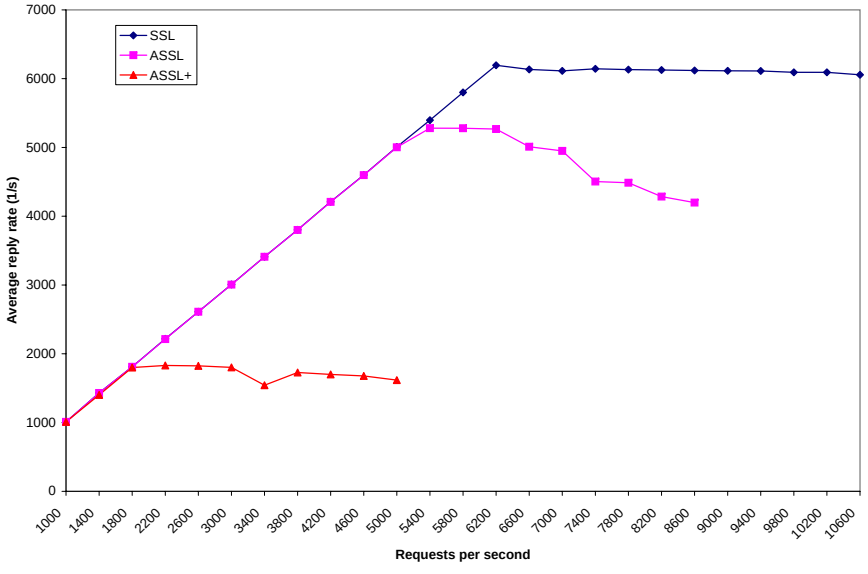


Figure 5.6: Performance Graph showing 1000 requests per session

where the clients establish few new SSL sessions making many requests per session and the other where the clients establish a new SSL session per request. In realistic situations the client behavior will be a mixture of these two extremes. We evaluate the overall effect of using ASSL in these extreme scenarios.

**Scenario 1.** In this experiment we evaluate ASSL's performance for the case where clients create minimal new SSL sessions when compared to the number of requests. To that end, 1000 requests are made per session and SSL sessions are reused. In this scenario the server spends proportionately more time checking for renegotiations than negotiating new connections. We note that this is due to the fact that for every SSL session there are many client requests, each of which requiring a check against the Request Filter for a potential security renegotiation. This should penalise ASSL's performance.

Figure 5.6 shows that when using ASSL, Apache experiences peak load between 30% and 85% of the maximum number of requests that SSL can support, depending on the Request Filter length and complexity. So, we indeed see that checking if

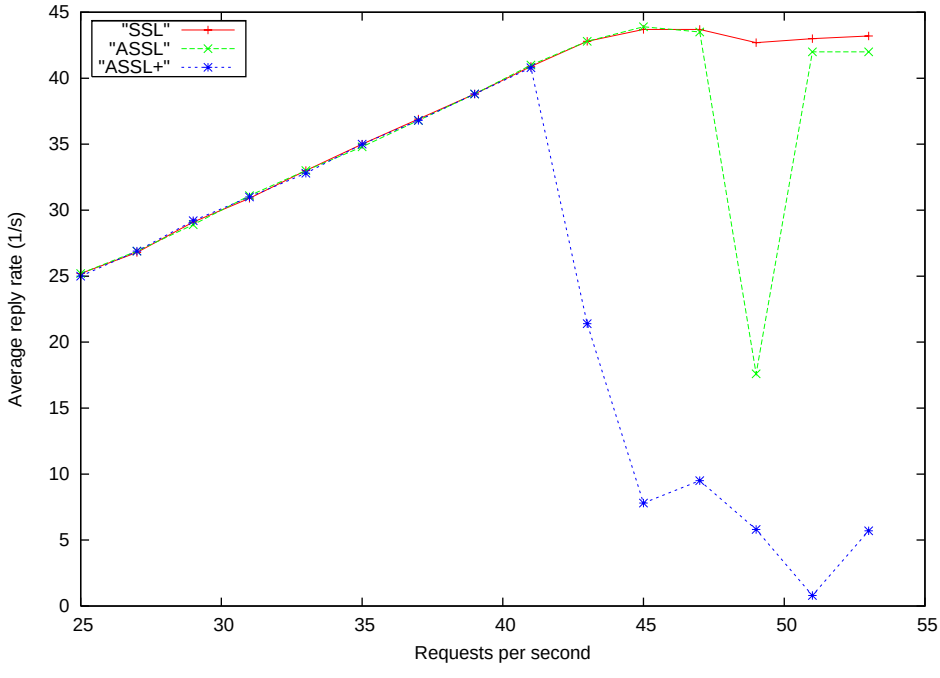


Figure 5.7: Performance Graph for one request per SSL session

renegotiation is required can be costly and will reduce the amount of clients that can be supported by a server. Note again that this client behavior represents an extreme case.

**Scenario 2.** In this experiment we evaluate ASSL's performance when clients create the maximum number of new SSL sessions, namely one session per request. In this scenario ASSL is only utilised once in each SSL session.

Figure 5.7 shows that even when using a full Request Filter the server incurs negligible additional overhead when using ASSL under normal client workloads. The performance does however break down when the server reaches peak load. We think this is because we hit another bottleneck. We observed that the number of child processes created by Apache rapidly increases for ASSL and ASSL+ under peak load. This is likely due to requests waiting for a lock on the Request Filter which results in Apache not releasing resources allocated to that request. Apache was configured with a theoretical limit of 40000 child processes and so we believe that the drop in

performance is likely due to the number of connections that can be supported by the system itself. A bespoke locking and threading solution could remedy the observed behavior but this is beyond the scope of our work.

Also note that increasing the requested file size, currently 44 bytes, will increase the cost of encryption for SSL as well as ASSL and so the additional cost associated with ASSL will be even smaller relative to the other server costs and so the performance difference between SSL and ASSL will be less significant. In general terms, the added functionality of being able to adapt the security level comes at a cost that only becomes prohibitive when the requested file size is small and the Request Filter is complex and executed often.

## 5.3 Summary

In this chapter we have shown how to effectively augment cryptographic security measures between a client and web server with adaptive security features which can respond to a variety of environmental influences at runtime in a timely and effective manner. This was achieved through the application of our methodology and adherence to the Adaptation Unit design in Section 4. Through the utilisation of adaptive design and software engineering principles in Chapter 2 we created a solution which can easily be adopted by both client and server. It enables extensive adaptation possibilities through the use of specialised $3rd$ parties and provides this in an efficient and performant manner. Experiment 1 showed that the amount of overhead introduced by ASSL is small (about 60 microseconds) but that the overhead is sensitive to the contents of the Request Filter (moving up to the order of milliseconds). When stressing the server (Experiment 2) much depends on the client behavior, which determines what proportion of processing time is spent executing the Request Filter. In particular, if load experienced by the server is due to an increase in new SSL sessions, i.e new clients arriving, an ASSL enabled server experiences little additional overhead. If however server load is experienced due to an increase in the number of requests per client, ASSL reduces the maximum server load buy 15% or more depending on Request Filter size.

# Chapter 6

# Security-performance trade-off

In Chapter 1 we considered the challenge of choosing the right level of security to protect a system from potential attackers. The decision has to incorporate a wide variety of factors from available processing and financial resources to more subjective factors such as the level of trust in the particular algorithms, expected cryptanalytic developments or the expected average future cost of computing resources to name but a few. Many of these factors are often ignored by system administrators and even when considered only allow them at best to make a "best effort" assessment based on subjective information. Even once a decision has been made the external factors on which it was based may also change. Examples include the value of the protected data or the threat level at a particular moment in time. Therefore making only one security decision can in itself leave the system at risk or break QoS guarantees.

One could envision an intelligent security system able to use additional resources, as and when they are available, to address this problem. To this end we chose to consider the web server environment as our problem domain, focusing on security guarantees provided through the Secure Socket Layer (SSL) protocol for client-server interactions. We have studied and evaluated current implementations of cryptographic algorithms used in SSL and have shown that the implementations themselves play a significant part in the expected algorithm performance. Due to the lack of Adaptive Security literature in the area we designed and implemented our own Adaptive SSL solution which adapts at runtime between these cryptographic algorithms. We now build on this research by formulating and demonstrating an effective security-performance trade-off.

In this chapter we apply the next three steps of our methodology. Since we have already chosen a crosscutting concern, i.e. system performance, we start by applying step three of the methodology in the System Analysis section (Section 6.1). In this section we identify measurable factors which influence the security-performance relationship. Through offline experimentation we study their impact under different client load conditions in Section 6.2. This in effect creates a representation of the security-performance interrelationship as detailed in the graphs in Section 6.3.5. The offline data is the Service States data as described in our Design, Section 4.2.

In the final section, Section 6.3, the last step of the methodology is applied. Though runtime measurement of the aforementioned cross-cutting concern and utilisation of the precomputed Service States we endeavour to reach our trade-off goal.

## 6.1 System Analysis

To trade off security and performance we first study their interrelationship. To better understand the performance impact of a security adaptation on the server we break down the SSL security cost into its constituent elements and identify the relevant server side SSL security mechanisms which contribute to the security cost. We secondly investigate how client behaviour affect the load each mechanism places on the server and how this influences the overall server load when an adaptation is actuated.

### 6.1.1 SSL costs

The first mechanism to consider is SSL session negotiation. Through this negotiation process each client establishes a secure SSL session with the server (as explained in Section 5.1). Negotiation occurs once per session and sessions are re-used for a number of future requests. The handshake protocol (Section 5.1.1.2) is expensive, involving the creation of the necessary key data as well as a number of client-server message exchanges.

The server also has to manage the security state of all current clients. This involves storing and retrieving relevant client information from the security cache when a request is made.

Table 6.1: Average number of bytes processed per second (rounded to the nearest 100).

| Cryptographic Algorithm | Throughput (kB/s) |
|---|---|
| RC4 | 112000 |
| AES 128 | 36000 |
| AES 192 | 30600 |
| AES 256 | 26500 |
| DES | 19300 |
| 3DES | 6900 |

Lastly the server is responsible for cryptography as it needs to decrypt client requests and encrypt the relevant responses.

Adapting the security, i.e. changing the security algorithm used, only has a server performance impact for encryption and decryption. It therefore follows that if the server is busy and it spends a large proportion of its processing resources on cryptography, changing the security would have a significant performance benefit. If however the server is busy but it spends more time managing the security and less time on cryptography then adapting the security would have less of a performance impact.

### 6.1.2 Client load patterns

In aid of identifying the actual relation between server load and the resource requirements of the server security mechanisms discussed in Section 6.1.1, we investigate the following aspects of client load patterns in the next section (Section 6.2):

**Security Algorithm** The performance cost incurred by the server through encryption and decryption is significantly influenced by the particular software implementation as we have shown in [90]. For our experiments we have chosen to use OpenSSL [86] version 0.9.8d. Table 6.1 shows the average measured throughput for each algorithm. Slower algorithms should result in lower server throughput. Table 6.1 shows that 3DES is the slowest algorithm and is about 16 times slower than RC4 which is the fastest. Care should be taken as slower algorithms do not necessarily provide more protection. This is highly implementation dependent. For instance in this version of OpenSSL (0.9.8d) AES provides more protection and outperforms DES whereas in OpenSSL version 0.9.7f it performs worse.

**Data size** The size of the requested data has a direct impact on the cryptography cost. The average file size is determined by the type of web site or e-commerce application accessed as well as client behavior.

**Session duration** The length of time clients spend using the service has a significant impact on the SSL management cost. Longer sessions results in more concurrent users which means larger security cache which leads to an increase in client status retrieval costs from the cache.

Each of the above client load patterns affect the proportion of resources allocated to each security mechanism and so influence the performance impact of adapting the security when the server is under load. We investigate the extent of such an impact through experimentation in the next section.

## 6.2 Experiments

Through experimentation this section will determine the performance impact of a security adaptation under the client load patterns identified in Section 6.1 and in so doing bring to light the interrelationship between performance and security. As we anticipated in Section 6.1.2 we will see that the level of cryptography, file size and session length play key roles and we will discover the extent of their impact.

All experiments were conducted on a 100Mb/s ethernet test bed of 11 identical Pentium III 866MHz machines with 512Mb RAM. Apache 2.2.3 and OpenSSL 0.9.8d toolkit was used.

Parameters chosen for the experiments are loosely based on the findings in [106]. Each experiment is based on a scenario where clients arrive at the server with increasing frequency, each client creating a new SSL session. Every client makes 64 consecutive requests at intervals of 4s average in a session. A client will wait no more than 6s for a reply after which it continues immediately with the next request.

For ease of reading, all figures show the load in requests per second (req/s) on the x-axis rather than in sessions/s. Response time (y-axis) represents the time between the client sending a request and receiving the response (encrytion/decryption time included).

In this section we first present the load generator we implemented to handle the adaptive nature of our experiments. Thereafter we consider to what extent client load patterns affect the performance cost or gain when a security adaptation is initiated.

### 6.2.1 Load Generator

Load generators simulate multiple client requests to a server. Due to the lack of SSL enabled load generating software that can handle an SSL adaptation, we built a custom tool utilising and extending Jakarta Commons' HttpClient 3.1 modules [107].

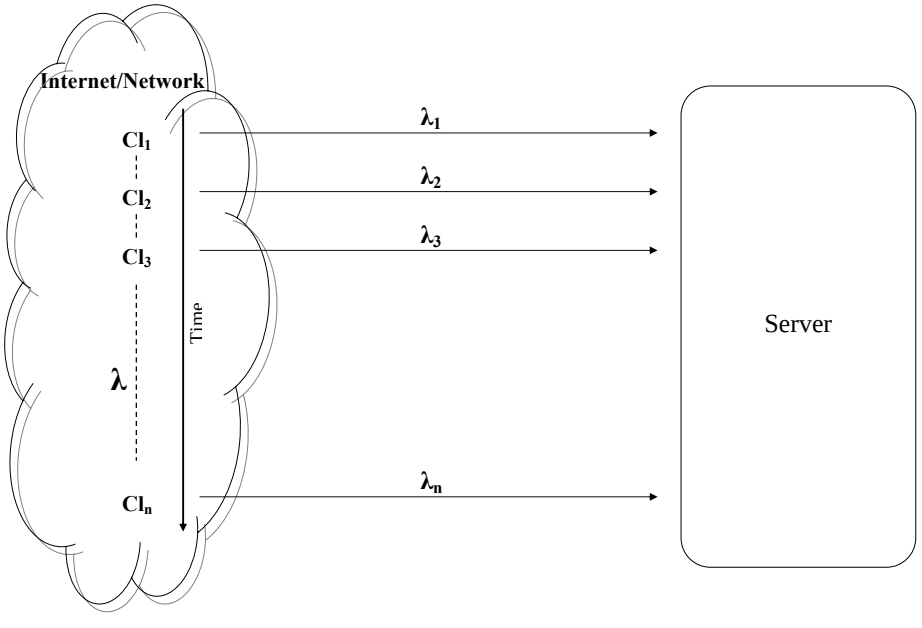


Figure 6.1: Load generator tool

As can be seen in Figure 6.1 the tool establishes new client sessions with the server at a certain rate ($\lambda$), following a Poisson process with some mean. Within each session the client can make multiple requests ($\lambda_n$), the average request rate also follows a Poisson process with some mean. The tool allows for both mean client arrival and request rates to be changed during an experiment to better simulate real client load. It also maintains a unique security state for each client session instead of sharing session information amongst clients. Many SSL enabled load generators share

this information for memory and performance reasons. Storing unique information for each client allows the server to adapt the security for all or only some clients, when the session is being established or during a session.

In addition to automated collection and graphing of results the tool provides a combination of features which are not provided as a feature set in other freeware software applications (features may be present in isolation in other load generating tools). The set of essential supported features are as follows. Firstly the SSL protocol is used to secure client sessions. Simulated clients are also divided amongst N physical machines to reduce the chance that the client machine becomes the bottleneck. (i.e. When results indicate that maximum server throughput is reached it is due to server- rather than client machine overload) The tool also allows SSL secured sessions to be renegotiated at runtime. Additionally, a unique SSL session state is used for each client rather than sharing SSL state information on the server side amongst sessions to reduce load on the server machine. This better simulates real client load on the server. I.e. the server has to manage a separate state for each client and so the load on the server more closely resembles a real world scenario. It also allows the server to initiate a SSL renegotiation for a subset of clients. It furthermore provides non-deterministic client arrival and request rates to better simulate real client load. Lastly, simulated clients are created using a low overhead threading mechanism to support high client arrival rates. This allows the clients to place the server under heavy load without the client machines becoming the bottleneck.

### 6.2.2 Security Algorithm

In this experiment we show the performance impact of the OpenSSL cryptography algorithms in Table 6.1. Requests are made for a 8192B file. Client arrival rate starts at 10 new clients every 2.5s (240 req/s), each client behaving as described above, and continues, decreasing delay between each batch of 10 client arrivals by 0.1s every time, until the server is overloaded. All response times under 250ms exhibit 90% confidence intervals of under 2ms and all response times over 250ms have 90% confidence intervals under 10ms.

Figure 6.2 shows that under increasing load the server can serve approximately

290 req/s using 3DES before it becomes overloaded. It can however serve 10%-30% more requests per second if it uses another algorithm. The figure shows that the server becomes overloaded in the order shown in Table 6.1, though for this particular file size and level of client concurrency RC4 only outperforms 3DES by 30% rather than the 16 fold increase shown in the Table 6.1. This is likely due to the fact that the resources the server has to delegate to the other tasks are relatively large per request compared to the resources allocated to cryptography and so the difference in cryptography performance is not so pronounced.

Nevertheless, we see clearly that the choice of encryption algorithm matters as it determines the maximum server throughout. Using a different level of security could therefore increase the number of clients supported by 10%-30%.

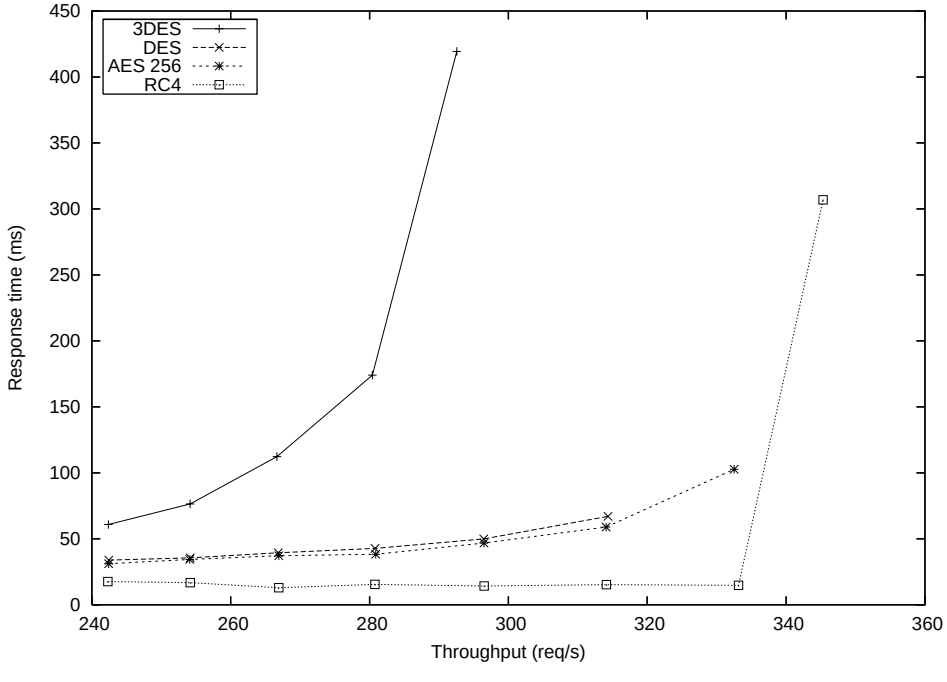


Figure 6.2: Cryptographic protocol overhead

### 6.2.3 Data size

This experiment shows the performance impact when requesting three different file sizes, 1024, 8192 and 12288 bytes. Client arrival rate starts at 10 new clients every 3.1s, decreasing delay between arrivals by 0.1s every time. All response times under 150ms exhibit 90% confidence intervals of under 2ms and all response times over 150ms have 90% confidence intervals under 5ms. The point at which the server overloads is the last plotted point on the figure, after which the response time shoots up.

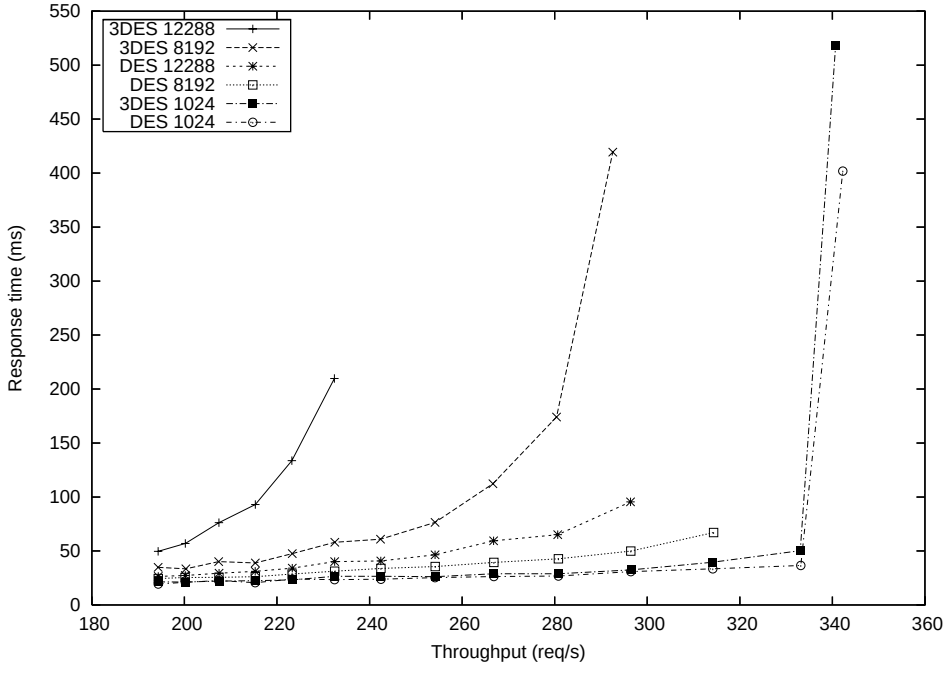


Figure 6.3: DES and 3DES performance under different file sizes

Figure 6.3 shows that when comparing DES and 3DES the difference in maximum throughput that that server can achieve grows to almost 30% as the file size increases. In other words, if clients request a small file size, such as 1024 bytes, the server gains almost no additional throughput if the security is adapted from 3DES to DES. This however increases to almost 30% as file size increases. For 12288 byte files the maximum server throughput increases by almost 30% from 232 req/s to 296 req/s

when security is adapted from 3DES to DES.

The figure also shows that considering requests per second only is not sufficient in deciding whether to adapt as the average requested file size plays a key role in its effect on the system load. For example; changing from DES to 3DES encryption at 242 req/s would overload the system if large files (12288B) are requested but would be relatively safe for files of 8192 bytes or less.

Since there is little difference in performance for smaller files, higher security should always be used in such cases. ASSL therefore has a more significant impact in scenarios where the client may be viewing or downloading larger files from a service. Examples could include online photo albums, music downloads, email attachments, etc.

### 6.2.4 Session duration

This experiment shows the performance impact of adapting security as client session time varies. 64 requests per session are made by all clients. Clients with short session lengths finish their 64 requests in 6.4 seconds (10 req/s) and clients with longer session lengths finish theirs in 256 seconds (0.25 req/s). Client arrival rate starts at 10 new clients every 3.1s and 8192B files are requested. All response times under 200ms exhibit 90% confidence intervals of under 2ms and all response times over 200ms have 90% confidence intervals under 5ms.

Figure 6.4 shows that adapting security from 3DES to DES for shorter sessions achieves a 35% throughput increase compared to 7% for longer sessions of 256s. Longer session durations mean more concurrent clients at the server which in turn results in more SSL management overhead. We therefore observe a smaller server performance impact due to cryptography adaptation.

In practice client behavior exhibits a mixture of session durations with the average duration dependant on the type of service hosted on the server.

ASSL shows promise in scenarios where server load is a result of frequent client-server interactions rather than SSL management costs due to long client sessions. Highly interactive web sites where clients are likely to finish quickly will therefore exhibit a large performance gain when adapting security using ASSL. Highly interactive

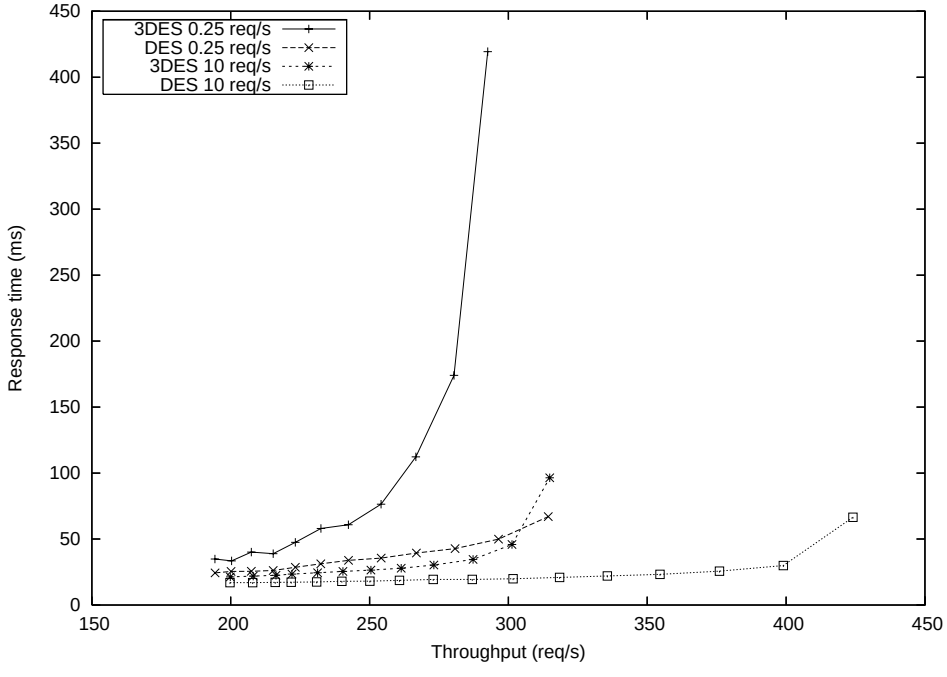


Figure 6.4: DES and 3DES performance with varying session duration

web pages are also becoming more prevalent on the web since the advent of Web 2.0.

It should also be noted that single server pages are often made up of various pieces of information such as images, style sheets, and scripts. Each client request therefore resulting in several additional automated requests per page. Sites where frequent client-server interactions are generated in this automated way will therefore also exhibit a larger performance gain.

## 6.3 Security-performance trade-off

Through a use-case scenario, based on the results obtained in Section 6.2, this section will trade off security and performance to achieve our trade-off goal. Namely, to show how Adaptive SSL can successfully maximise security by utilising available processing resources whilst still respecting client QoS requirements.

To achieve this we implement our Adaptive Security Service design as detailed in Section 4.2.

The first component to consider is the Monitor Unit which collects data relating to the current system resources. Results are passed to the Trade-off Engine where adaptation policies determine the performance cost/benefit of a security adaptation based on the collected Service States information. Security is adapted through the Adaptation Unit.

The ASSL module represents the Adaptation Unit allowing effective security adaptation at runtime. ASSL does not stipulate its intended use and so we are able to extend the solution with our own specialised $3^{rd}$ party software and complete the feedback loop.

In this chapter we first present the scenario to which our adaptive system will react. We demonstrate how a non-adaptive system responds in this scenario. We then consider two adaptation policies based on our results from Section 6.2 for the Trade-off Engine. The chosen policy is implemented as part of our specialised $3^{rd}$ party software and employed in the scenario to demonstrate its practical utility. The Monitor Unit is also part of our $3^{rd}$ party implementation though much responsibility is delegated to specialised system level monitoring software.

### 6.3.1 Use-case scenario

The use-case will depict a scenario where a server experiences a sudden influx of client arrivals. The server has three available cryptography algorithms. In increasing order of security and performance cost they are RC4, DES and 3DES. For this scenario RC4 is considered adequate to protect the available data, though as discussed in Chapter 1 a security threat inevitably remains. Clients also expect a certain level of Quality of Service in that they will not wait indefinitely for a server response.

We will first show how the client arrival influx affects the server when using each security level. We then demonstrate how ASSL can improve on the required security level (i.e. RC4) whilst respecting client QoS requirements.

### 6.3.2 Experiment setup

For this use-case the client behavior is as follows. Each client has an average session duration of 256 seconds consisting of 64 requests at 4 second intervals. Each request

is for a 12288B file.

For the first 500s 10 clients arrive every 3s (combined average request rate, as perceived by the server, is 200 req/s). The next 250s client arrivals increase to 10 clients every 2.5s (245 req/s) and then decreases back to 200 req/s for the last 500s. Each of the throughput values on the figure is a throughput average over a 10s interval. The Poisson client arrival and request processes are based on a fixed set of random number generated seed values, and so the figures show a particular experiment trace. The particular trace however has little impact on the average throughput every 10s due to the large number of requests in each 10s interval. Experiments were also done multiple times with different seeds to calculate the average number of timed out requests, i.e. those requests that broke the client QoS requirements. Other details on the experimental setup can be found in Section 6.2.

We will show that both RC4 and DES can cope under the client load influx whereas the more secure 3DES can not. We will also show that ASSL can support the client load and additionally use available server resources to increase the security.

### 6.3.3 Basic security (RC4)

Figure 6.5 shows the server throughput under client behavior described above when using RC4 encryption. The figure depicts a server that can cope under client load. Additional results affirm this as no client requests timed out (i.e. all replies were received within 6s. See Section 6.2). Repeating the experiment a number of times with different seed values also produced no timeouts.

In Figure 6.5 clients start to arrive in batches of 10 every 3s for the first 500s. After 256s clients also start to leave the system (they have completed their 64 requests) and the server throughput stabilises at 200 req/s. The second (245 req/s for 250s) and third (200 req/s for 500s) phase of the experiment show the client arrival influx and also behave as expected.

The server throughput figure for DES is similar to Figure 6.5 albeit at a higher CPU load. We can see in Figure 6.3 that DES can also sustain a request rate of 245 req/s whilst respecting the 6s client QoS constraint. No client timeouts were recorded for DES.

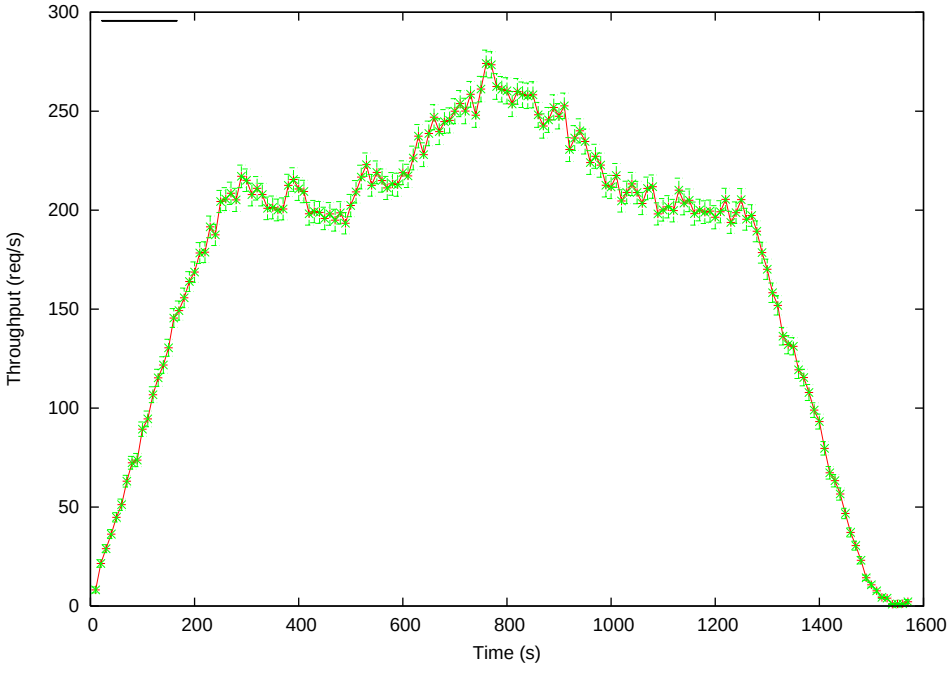


Figure 6.5: Server throughput using RC4

### 6.3.4 High security (3DES)

Figure 6.6 shows the server throughput when using 3DES under the same client behavior as above. The bar graph additionally shows the percentage of requests which timeout in each interval. It depicts a scenario where the server does not cope under client load. We can also see this in Figure 6.3 where the server overloads before 245 req/s when using 3DES. From Figure 6.6 we can see that requests start to timeout 500s into the experiment when the client load increases beyond the level at which the server can cope. Because clients only submit the next request after receiving a response for the previous one, clients make fewer requests per second as they have to wait longer for a reply and so server throughput drops. Due to the slower request rate the sessions also become longer and so as more and more clients arrive at this higher rate the number of concurrent sessions also increase. More concurrent sessions results in an increase in the total request rate, although nearly all of the requests time out. Even when no more new clients arrive after 1250s, and current

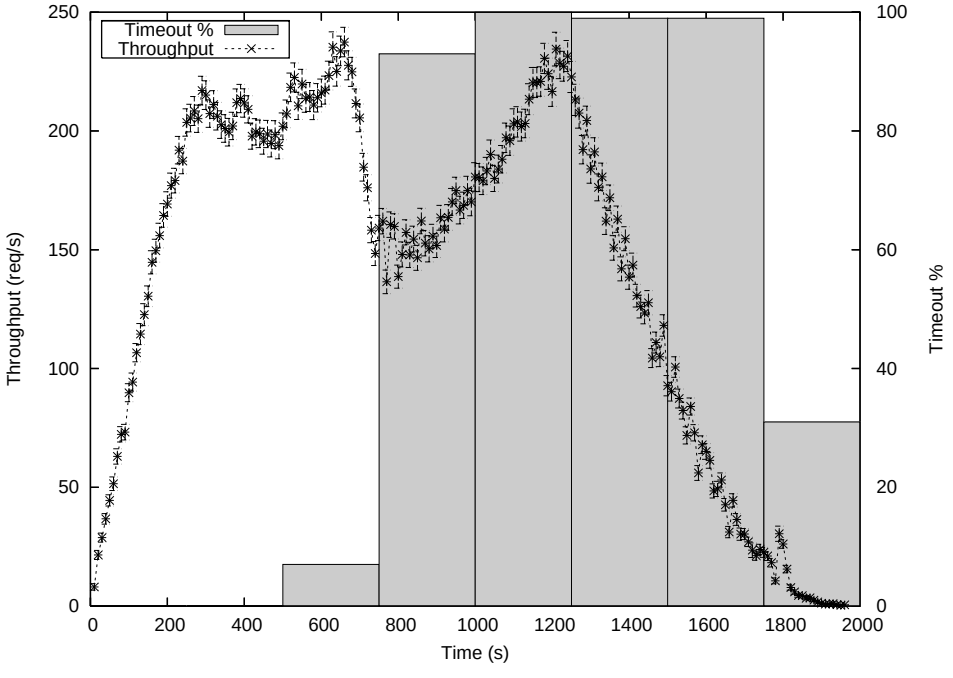


Figure 6.6: Server throughput using 3DES

clients finish their sessions, most of the requests still time out. This is likely due to the fact that the server is still buffering and waiting to serve old client requests, as it has no way of knowing that the client has timed out, and so the new requests still have to wait until the buffers have cleared by which time they might also have timed out.

For 3DES in total 144642 client requests timed out. Repeating the experiment 6 times with different seed values showed an average timeout of 148737 with a deviation of 1444 at 90% confidence interval.

This experiment showed how performance degrades in a standard non-adaptive server when an influx of client arrivals overload the server.

### 6.3.5 Adaptive security

This section will show how Adaptive SSL can be utilised in this scenario to trade off security and performance by using the available processing resources to increase security whilst still respecting client QoS requirements. We implement our own spe-

cialised $3^{rd}$ party application to monitor system load and make adaptation decisions. It adapts the security by interacting with our Adaptive SSL solution as discussed in Chapter 5.

Based on the client load patterns studied we develop two policies to trade off security and performance. One through direct and the other through indirect monitoring of the server throughput. The most promising policy candidate is implemented as part of our $3^{rd}$ party application and shown through experimentation to be an effective and robust solution.

#### 6.3.5.1 Throughput policy

This policy depends on direct monitoring of the server throughput. In particular we consider monitoring throughput with a view to adapt the security based on the current average requested file size. We extend the experiments in Section 6.2.3 with results for file sizes 16384 and 20480 bytes and plot only the maximum achievable server throughput in Figure 6.7.

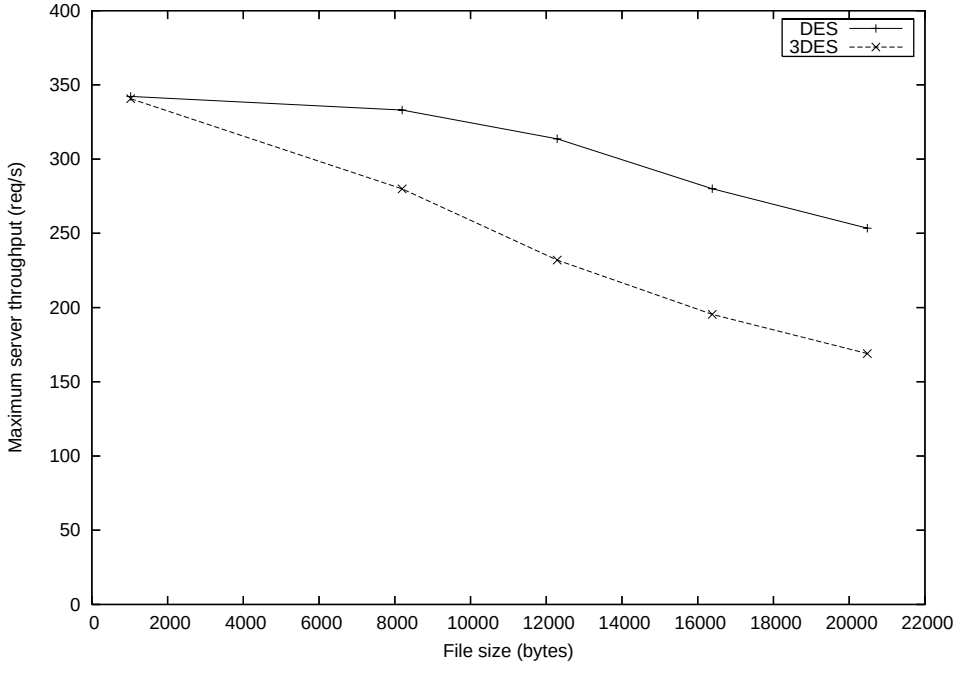


Figure 6.7: Peak server throughput for DES and 3DES under different file sizes

The figure shows that the difference in peak server throughput between DES and 3DES remains constant for files sizes greater than 12288 bytes. This is an unexpected result for which we cannot account but which consistently shows up in our experiments. One would expect the difference in peak performance to continue to diverge.

From the figure we derive two approximate functions for DES and 3DES using the last two points plotted. We note that this is only an approximation and can be derived in any number of different ways.

$$f_{des}(x) = -\frac{13x}{2048} + 384$$

$$f_{3des}(x) = -\frac{13x}{2048} + 299$$

The policy utilises the functions to adapt security at runtime as follows. Let T be the current server throughput, $x$ the average requested file size and *enc* the current security algorithm. *enc+1* and *enc-1* denotes an increase and decrease in security respectively.

$$\begin{aligned}
&\text{If T} \geq f_{enc}(x) \text{ then} \\
&\quad f_{enc-1}(x) \\
&\text{Else if T} < f_{enc+1}(x) \text{ then} \\
&\quad f_{enc+1}(x)
\end{aligned}$$

The policy reads as follows. If the current server throughput T reaches the peak throughput the server can maintain under security level *enc* and average file size $x$ then decrease the security. If however the current server throughput T is low enough to increase the security then do so. It would also be prudent to adapt the security before the maximum throughput is reached. Examples could include defining a throughput range before the maximum throughput in which the security can be adapted or a more advanced technique by which the expected throughput for the next throughput measurement is predicted based on the current throughput rate of change. This is however beyond the scope of this thesis.

It should be noted that although file size is considered here the average session length also plays a key role. Similar graphs and policies can be formulated to consider variations in session length.

One major concern of measuring server throughput directly is the monitoring cost. Monitoring itself could overload the system or negate performance benefits achieved through adaptation [17]. We consider an alternative in the next section.

#### 6.3.5.2 CPU load policy

This policy depends on indirect monitoring of the server throughput by recognising that increased throughput implies increased server CPU load. The operating system already monitors CPU load as a background process. This information can be utilised cheaply by our adaptive policy rather than implementing a bespoke application level throughput monitoring system which may adversely affect the maximum throughput results obtained thus far.

In order to utilise the available resources one needs to know how much of it is free as well as how much will be required or freed when the security is adapted.

Figure 6.8 shows the server performance implications for the three levels of encryption when clients have an average session duration of 256s and request files of 12288 bytes each. The figure shows the average CPU load at different client arrival rates, starting with 10 clients every 3.1s (i.e. total number of requests as perceived by the server is 195 req/s) and reducing the delay between arrivals by 0.1s each time until the server is overloaded. CPU % is the average CPU load measured using iostat [108]. 100% implies that the CPU is fully utilised and potentially have further idle tasks waiting to use the CPU.

The figure shows that using 3DES the server overloads just after 230 req/s with DES and RC4 just after 296 req/s (not shown). This affirms the results in Figure 6.3 and provides further insight. The figure also shows that a server with load over 70% and using DES should not consider adapting to 3DES because a server under the same client throughput and using 3DES would be overloaded. This is also true of moving from RC4 to DES over 50% and from RC4 to 3DES over 40%. Decreasing the security is also prudent when the CPU utilisation reaches 100%.

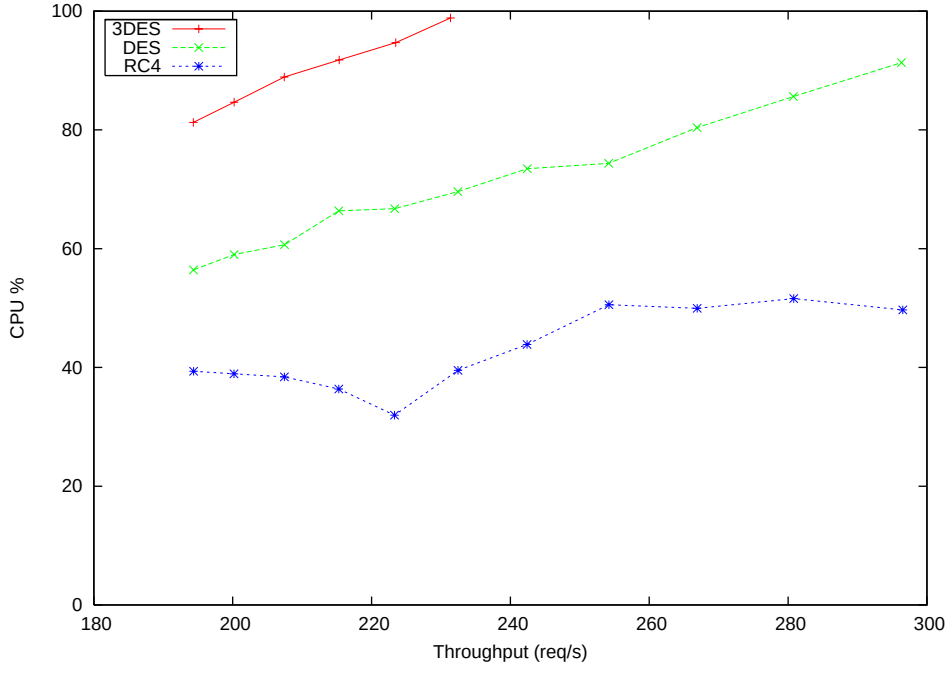


Figure 6.8: CPU utilisation under different client loads

Using the insights gained above we provide an adaptation policy in Table 6.2 that will more effectively utilise the available resources to improve security without overloading the server and so break client QoS requirements.

Table 6.2: Adaptive SSL policy reference.

| | New protocol | | |
|---|---|---|---|
| Current protocol | $3DES$ | $DES$ | $RC4$ |
| 3DES | X | 100 | 100 |
| DES | 70 | X | 100 |
| RC4 | 40 | 50 | X |

Pseudo code for the adaptation algorithm is provided below. It compares the current CPU level with the values in Table 6.2 and determines whether the current level of security should be changed. The code first determines the current security protocol in the first column of the table and moves to position X in that row. It then considers increasing the security by comparing the current CPU load with all the values to the left in that row. If the current CPU level is less than the particular value security can be increased to the algorithm that column represents. For example, if the current security is DES we move to the X in row 2 column 2. The current CPU load is compared to the value in the 3DES column to the left. If CPU load is low enough for an increase in security, i.e. CPU $< 70\%$, then 3DES is selected. The same is done for values to the right (using the $>=$ operator) to determine if security should rather be decreased.

```
#comment
load = getCpuLoad
PolicyTable(row, column)
curSec   #current Security level
newSec   #new Security level

#Move to X
Move to PolicyTable(curSec,curSec)
   #Check if security should be increased
   #Check all values to the left
   IF load < PolicyTable(curSec, curSec - n)
     RECORD newSec
```

```
    #Check if security should be decreased
    #Check all values to the right
    IF load >= PolicyTable(CurSec, CurSec + n)
      RECORD newSec

IF newSec THEN adapt
```

Monitoring costs are a known performance concern for many systems [17]. We we seek to avoid this by utilising a policy based on CPU load in the next section.

#### 6.3.5.3 Adaptive Experiment

For this experiment the CPU was monitored during the use-case scenario using iostat [108]. Every 10s our CPU monitoring program computes the average CPU load recorded over the previous 10s by iostat and adapts the ASSL security based on the CPU policy in Section 6.3.5.2.

Table 6.3: Adaptive SSL security adaptations.

| New security | At time (s) |
|---|---|
| 3DES | 0 |
| DES | 450 |
| RC4 | 460 |
| DES | 720 |
| 3DES | 780 |

Table 6.3 shows when security was adapted in Figure 6.9. The values show that through utilising ASSL the CPU monitor was able to effectively maximise security for most of the experiment duration. All new clients arriving before 460s and all new clients arriving after 720s received better than the recommended security level.

It also did this whilst maintaining the client QoS requirements. Figure 6.9 depicts a scenario where the server can cope under the client load. In total, only 8 client requests timed out. From the figure we can see that the CPU load was effectively reduced during the client influx at around 500s by decreasing the security level to DES and then to RC4 (see Table 6.3) and so preventing the server from overloading and respecting the client QoS requirements. Once the server load decreased to a safe level below 50% (see Table 6.2) the security was increased to DES and then to 3DES

in the same manner. The figure thus shows the server CPU load increasing again after approximately 750s.

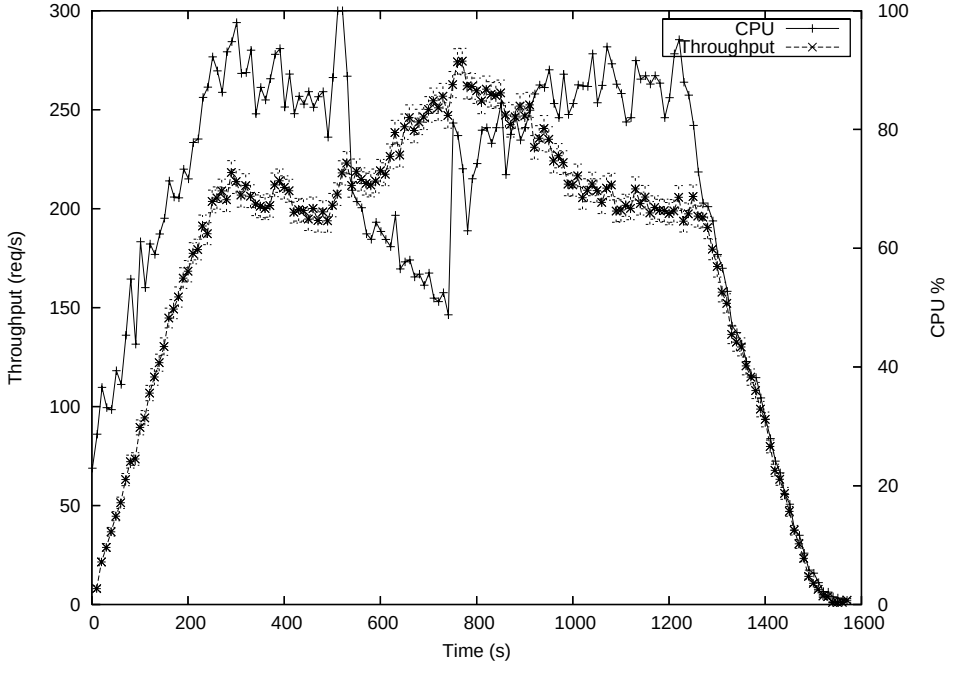


Figure 6.9: Server throughput under ASSL security

We observe that the security was reduced twice in a short period of time (once at 450s and again at 460s) and also increased rapidly in a short interval (once at 720s and again at 780s). This is due to the fact that the first security change has perhaps not had time to make a significant enough impact on the server load before the next chance for adaptation occurs. Of course if the time between adaptations is increased and the increase in arrival rate is large enough for the first adaptation to make little impact then the server would become overloaded. Choices on how frequently an average should be taken and how many seconds an average should account for is a needs based decision and dependant on the environment the server is deployed in.

Drawing from the details in this section it should be noted that our adaptation policy is very robust to client behavior. Firstly, security is reduced as soon as the server reaches its maximum load. This is due to the short CPU sample times (10s)

as well as the fact that our policy does not wait to determine if the load might decrease or stabilise but reduces the security immediately. Secondly, security is only increased if the server could cope with the same load at a higher security level. When security is adapted only new clients start with the higher level of security and so it takes some time before the server is serving all clients at this new level. Our policy is therefore very cautious with respect to increasing security and potentially overloading the system or breaking client QoS.

## 6.4 Conclusion

In this chapter we showed the benefits of applying our methodology. We analysed the server performance cost resulting from an SSL security adaption under different client loads and behaviours and showed how this information can be successfully exploited to predict the future system state when deciding whether to adapt the security. We demonstated that this information can successfully be used to create intelligent adaptation policies.

We have shown that the average requested file size and the SSL session duration has a significant influence on the performance impact of a security adaptation and so should be considered in addition to the server processing load or client throughput. Adapting security has a greater server performance impact when requested files are large and when clients make multiple short requests. Consequently the server may choose to provide better security for smaller files and clients with longer sessions or less security if files become prohibitively large and sessions are short.

We showed how server throughput and client QoS can degrade during an influx of client arrivals if the default security level is high in a standard non-adaptive server. We demonstrated that ASSL can effectively allow our $3^{rd}$ party software to adapt SSL security at runtime in a real system. We also formulated a robust adaptation policy and showed that it can provide better security by utilising available resources.

# Chapter 7

# Conclusion

In this thesis we recognise that achieving complete "separation of concerns" between SSL security and performance is merely a fool's hope. To resolve this contention we deferred the trade off from design time to runtime, thus allowing an informed trade off policy specification based on up to date contextual information only available at runtime. To that end we specified our goal as follows: To *"Demonstrate the practical utility of Adaptive Security in trading off security and QoS at runtime."*

We first provide an overview of the thesis, reflecting on the principles and key lessons learnt in Section 7.1. We conclude the chapter and the thesis with insights into potential future research directions.

## 7.1 Principles & Key lessons

The underlying principal focus and driving force in this thesis is the desire to move security decisions from a design or deployment time decision to that of a runtime decision. Many systems, architectures and frameworks currently exist to support such runtime decisions but the challenge we address in this thesis lies in *moving* such a decision from a static design/deployment time decision to a dynamic runtime decision. This concept is at the heart of our methodology which takes legacy security systems and provides steps to realise an adaptive security system.

When implementing the methodology a number of key lessons have been learnt which provide further insight to our core contributions:

Firstly, we have found that an effective methodology must fully realise the auto-

nomic feedback loop mechanism. In so doing the methodology fully considers the significant facets of an adaptive system and as an end result produces an adaptive security system where the adaptation is not an emergent property of the system but explicitly represented in the system structure. The methodology does not encourage changing the existing system but instead advocates building on top of and around the existing system such that the legacy system becomes in effect only one phase of the adaptive feedback loop.

Secondly, the significant value of offline data in making complex runtime decisions. Such data is not only practical in that these potentially resource intensive processes are used to compute data offline but also allows for more complex runtime decision making. It allows for the offline study of the interrelationship between security and the crosscutting concern so that complex considerations regarding the future state of the system can then be made at runtime *before* a security decision is taken. This offline data lies at the core of our intelligent adaptive policy contributions which not only take into account the effect previous security adaptations have on the crosscutting concern but also predicts the future impact of security adaptations.

There are also a number of design and implementation challenges in following the methodology. A noteworthy part of this contribution is the performant nature of Adaptive SSL and the extended infrastructure. Having chosen performance as the crosscutting concern places additional requirements on the system in supporting adaptation without adversely affecting the crosscutting concern. The adaptation decision itself is however only part of the performance cost. Due to the feedback loop nature of the design and the emphasis placed on building on top of and around the existing legacy system one quickly accumulates additional components which all need to share in the processing resources. We note that the monitoring component often plays a significant role in these costs. An additional implementation challenge which must be met in following the methodology was to gain control of the security decision making process in the legacy system. Accomplishing this and providing a performant implementation of the design led to the achievement of our goal, namely "Demonstrating the practical utility of Adaptive Security in trading off security and QoS at runtime"

## 7.2 Future direction

In light of the diverse scope and novel nature of our research area there are a plethora of potential research directions to explore. We highlight the key areas we find of particular interest.

Firstly we consider experimental extensions of our research contribution. One such area already mentioned relates to when the adaptation occurs. In this thesis we adapted security when the server reached overload for a particular encryption algorithm, file size and session duration combination. It may however be prudent to adapt the security before an overload level is reached. One could consider defining a range of values within which to adapt the security before overload occurs such that if *currentLoad* $\geq$ *overload* - $n$ security must be adapted. Finding a suitable value for $n$ would be the objective in this case. One may also consider predicting the next server load measurement based on the current request rate increase or decrease and so adapt the security preemptively. i.e. if expected load $l \geq r \times t$ (where $r$ is the change in client request rate and $t$ the time at which the new measurement will be taken) then the security must be adapted. Logging server load over a number of weeks to establish expected client behaviour may also prove useful in preemptively adapting security. Further analysis of the adaptation process itself could also prove insightful. Various adaptation heuristics could be explored in understanding the performance effect of a security adaptation; adapting security for only a percentage of the clients may be appropriate. One can also consider a scenario where clients with different security levels are using the system and adapting the security for only certain groups of clients may achieve the desired performance goal. The total server processing cost for adaptation can also vary depending on the definition of adaptation. Adaptation may imply changing server security for all new clients, changing security for all clients at any particular moment in time or re-authenticating clients. Each method varying in processing and bandwidth overhead.

Secondly an alternate approach to the design and development of Adaptive SSL can also be considered. A design which additionally utilises client side code presents an opportunity to use multiple SSL sessions per client. Such concurrent sessions can rep-

resent different security levels and adapting security implies sending data over a more or less secure session. We therefore avoid the need for session renegotiation. Data units (e.g. a single web page) can also be fragmented into sections, sent over multiple SSL sessions and reconstructed on the client/server side. This has the additional cost of maintaining multiple concurrent SSL sessions per client and reconstruction of data units but allows for immediate added protection for particular sections that require it as well as adding minimal performance overhead during adaptation. A policy design that does not depend on pre-computed offline measurements may also prove useful. To that end one could adapt Queueing theory models for multiple job types to model and predict adaptation benefits.

Finally we consider research endeavours in the general area of Adaptive Security. To effectively evaluate security in an automated fashion it must be quantified. "Can we Quantitatively Assess Security?" [109] provides an overview of such research ranging from evaluating cryptographic key sizes [15], as is used in this thesis, to using Markovian theory [110]. An interesting extension of our work would be to quantify security in such a way that it can be directly traded off with performance using some utility function. Current literature suggests that inspiration for such a trade off might be found in the area of control engineering [17].

An Adaptive Security system that can take all security concerns into account, evaluate system Quality of Service and trade them off without human interaction still requires a considerable amount of active research. One fact however is slowly being accepted by the research community, and to this our research attests, security is not... **absolute**.

# Appendix A

# SSLRequire Directive [1]

The SSLRequire directive is an arbitrarily complex boolean expression specifying, in the case of ASSL, a condition for security re-/negotiation. The expression matches the following Backus-Naur Form syntax:

```
expr     ::= "true" | "false"
           | "" expr!
           | expr "&&" expr
           | expr "||" expr
           | "(" expr ")"
           | comp
comp     ::= word "==" word | word "eq" word
           | word "!=" word | word "ne" word
           | word "<"  word | word "lt" word
           | word "<=" word | word "le" word
           | word ">"  word | word "gt" word
           | word ">=" word | word "ge" word
           | word "in" "{" wordlist "}"
           | word "=~" regex
           | word "!~" regex
wordlist ::= word
           | wordlist "," word
word     ::= digit
           | cstring
           | variable
           | function
digit    ::= [0-9]+ cstring  ::= "..."
```

```
variable ::= "%{" varname "}"
function ::= funcname "(" funcargs ")"
```

... where *funcname* is the function *file(filename)* which takes a string argument and expands the contents of the file. *varname* represents any of the SSL variables in Table A.1 or CGI and Apache variables below.

Standard CGI/1.0 and Apache variables:

| | | |
|---|---|---|
| HTTP_USER_AGENT | PATH_INFO | AUTH_TYPE |
| HTTP_REFERER | QUERY_STRING | SERVER_SOFTWARE |
| HTTP_COOKIE | REMOTE_HOST | API_VERSION |
| HTTP_FORWARDED | REMOTE_IDENT | TIME_YEAR |
| HTTP_HOST | IS_SUBREQ | TIME_MON |
| HTTP_PROXY_CONNECTION | DOCUMENT_ROOT | TIME_DAY |
| HTTP_ACCEPT | SERVER_ADMIN | TIME_HOUR |
| HTTP:headername | SERVER_NAME | TIME_MIN |
| THE_REQUEST | SERVER_PORT | TIME_SEC |
| REQUEST_METHOD | SERVER_PROTOCOL | TIME_WDAY |
| REQUEST_SCHEME | REMOTE_ADDR | TIME |
| REQUEST_URI | REMOTE_USER | ENV:variablename |
| REQUEST_FILENAME | | |

Table A.1: SSL variables [111]

| Variable Name | Value Type | Description |
|---|---|---|
| HTTPS | flag | HTTPS is being used |
| SSL_PROTOCOL | string | The SSL protocol version (SSLv2, SSLv3, TLSv1) |
| SSL_SESSION_ID | string | The hex-encoded SSL session id |
| SSL_CIPHER | string | The cipher specification name |
| SSL_CIPHER_EXPORT | string | true if cipher is an export cipher |
| SSL_CIPHER_USEKEYSIZE | number | Number of cipher bits (actually used) |
| SSL_CIPHER_ALGKEYSIZE | number | Number of cipher bits (possible) |
| SSL_VERSION_INTERFACE | string | The mod_ssl program version |
| SSL_VERSION_LIBRARY | string | The OpenSSL program version |
| SSL_CLIENT_M_VERSION | string | The version of the client certificate |
| SSL_CLIENT_M_SERIAL | string | The serial of the client certificate |
| SSL_CLIENT_S_DN | string | Subject DN in client's certificate |
| SSL_CLIENT_S_DN_x509 | string | Component of client's Subject DN |
| SSL_CLIENT_I_DN | string | Issuer DN of client's certificate |
| SSL_CLIENT_I_DN_x509 | string | Component of client's Issuer DN |
| SSL_CLIENT_V_START | string | Validity of client's certificate (start time) |
| SSL_CLIENT_V_END | string | Validity of client's certificate (end time) |
| SSL_CLIENT_A_SIG | string | Algorithm used for the signature of client's certificate |
| SSL_CLIENT_A_KEY | string | Algorithm used for the public key of client's certificate |
| SSL_CLIENT_CERT | string | PEM-encoded client certificate |
| SSL_CLIENT_CERT_CHAIN | string | PEM-encoded certificates in client certificate chain |
| SSL_CLIENT_VERIFY | string | NONE, SUCCESS, GENEROUS or FAILED:reason |
| SSL_SERVER_M_VERSION | string | The version of the server certificate |
| SSL_SERVER_M_SERIAL | string | The serial of the server certificate |
| SSL_SERVER_S_DN | string | Subject DN in server's certificate |
| SSL_SERVER_S_DN_x509 | string | Component of server's Subject DN |
| SSL_SERVER_I_DN | string | Issuer DN of server's certificate |
| SSL_SERVER_I_DN_x509 | string | Component of server's Issuer DN |
| SSL_SERVER_V_START | string | Validity of server's certificate (start time) |
| SSL_SERVER_V_END | string | Validity of server's certificate (end time) |
| SSL_SERVER_A_SIG | string | Algorithm used for the signature of server's certificate |
| SSL_SERVER_A_KEY | string | Algorithm used for the public key of server's certificate |
| SSL_SERVER_CERT | string | PEM-encoded server certificate |

# Appendix B

# SSLCipherSuite Directive [2]

SSLCipherSuite directive is a colon separated list, in preferential order, of all algorithm combinations (or ciphers) to be considered for an SSL or ASSL security re-/negotiation. The ciphers may be listed separately as in Table B.2 and B.3 or specified using aliases listed in Table B.1. A combination of elements in the two tables may also be used and can be manipulated with the following prefixes:

- `No prefix:` add cipher to list
- `+` add ciphers to list and pull them to current location in list
- `-` remove cipher from list (can be added later again)
- ! kill cipher from list completely (can not be added later again)

For example; The SSLCipherSuite "ALL:!ADH:RC4+SHA:+MEDIUM:+SSLv2" indicates that all ciphers in Table B.3 should be included with the exception of those containing anonymous key exchange algorithms. Ciphers that use RC4 and SHA should be listed next followed by all ciphers that use 128 bit encryption (i.e. MEDIUM) and then all SSLv2 ciphers.

Table B.1: SSL alias list [2]

| Alias | Description |
|---|---|
| ALL | all SSL ciphers |
| SSLv2 | all SSL version 2.0 ciphers |
| SSLv3 | all SSL version 3.0 ciphers |
| TLSv1 | all TLS version 1.0 ciphers |
| EXP | all export ciphers |
| EXPORT40 | all 40-bit export ciphers only |
| EXPORT56 | all 56-bit export ciphers only |
| LOW | all low strength ciphers (no export, single DES) |
| MEDIUM | all ciphers with 128 bit encryption |
| HIGH | all ciphers using Triple-DES |
| RSA | all ciphers using RSA key exchange |
| DH | all ciphers using Diffie-Hellman key exchange |
| EDH | all ciphers using Ephemeral Diffie-Hellman key exchange |
| ADH | all ciphers using Anonymous Diffie-Hellman key exchange |
| DSS | all ciphers using DSS authentication |
| NULL | all ciphers using no encryption |

Table B.2: SSL algorithm list [2]

| Tag | Description |
|---|---|
| kRSA | RSA key exchange |
| kDHr | Diffie-Hellman key exchange with RSA key |
| kDHd | Diffie-Hellman key exchange with DSA key |
| kEDH | Ephemeral (temp.key) Diffie-Hellman key exchange (no cert) |
| aNULL | No authentication |
| aRSA | RSA authentication |
| aDSS | DSS authentication |
| aDH | Diffie-Hellman authentication |
| eNULL | No encryption |
| DES | DES encryption |
| 3DES | Triple-DES encryption |
| RC4 | RC4 encryption |
| RC2 | RC2 encryption |
| IDEA | IDEA encryption |
| MD5 | MD5 hash function |
| SHA1 | SHA1 hash function |
| SHA | SHA hash function |

Table B.3: Example of SSL Algorithm list [2]

| Cipher-Tag | Protocol | Key Exchange | Authentication | Encryption | MAC |
|---|---|---|---|---|---|
| DES-CBC3-SHA | SSLv3 | RSA | RSA | 3DES(168) | SHA1 |
| DES-CBC3-MD5 | SSLv2 | RSA | RSA | 3DES(168) | MD5 |
| IDEA-CBC-SHA | SSLv3 | RSA | RSA | IDEA(128) | SHA1 |
| RC4-SHA | SSLv3 RSA | RSA | RC4(128) | SHA1 | |
| RC4-MD5 | SSLv3 RSA | RSA | RC4(128) | MD5 | |
| IDEA-CBC-MD5 | SSLv2 | RSA | RSA | IDEA(128) | MD5 |
| RC2-CBC-MD5 | SSLv2 | RSA | RSA | RC2(128) | MD5 |
| RC4-MD5 | SSLv2 RSA | RSA | RC4(128) | MD5 | |
| DES-CBC-SHA | SSLv3 | RSA | RSA | DES(56) | SHA1 |
| RC4-64-MD5 | SSLv2 RSA | RSA | RC4(64) | MD5 | |
| DES-CBC-MD5 | SSLv2 | RSA | RSA | DES(56) | MD5 |
| EXP-DES-CBC-SHA | SSLv3 | RSA(512) | RSA | DES(40) | SHA1 |
| EXP-RC2-CBC-MD5 | SSLv3 | RSA(512) | RSA | RC2(40) | MD5 |
| EXP-RC4-MD5 | SSLv3 | RSA(512) | RSA | RC4(40) | MD5 |
| EXP-RC2-CBC-MD5 | SSLv2 | RSA(512) | RSA | RC2(40) | MD5 |
| EXP-RC4-MD5 | SSLv2 | RSA(512) | RSA | RC4(40) | MD5 |
| NULL-SHA | SSLv3 RSA | RSA | None | SHA1 | |
| NULL-MD5 | SSLv3 RSA | RSA | None | MD5 | |
| ADH-DES-CBC3-SHA | SSLv3 | DH | None | 3DES(168) | SHA1 |
| ADH-DES-CBC-SHA | SSLv3 | DH | None | DES(56) | SHA1 |
| ADH-RC4-MD5 | SSLv3 | DH | None | RC4(128) | MD5 |
| EDH-RSA-DES-CBC3-SHA | SSLv3 | DH | RSA | 3DES(168) | SHA1 |
| EDH-DSS-DES-CBC3-SHA | SSLv3 | DH | DSS | 3DES(168) | SHA1 |
| EDH-RSA-DES-CBC-SHA | SSLv3 | DH | RSA | DES(56) | SHA1 |
| EDH-DSS-DES-CBC-SHA | SSLv3 | DH | DSS | DES(56) | SHA1 |
| EXP-EDH-RSA-DES-CBC-SHA | SSLv3 | DH(512) | RSA | DES(40) | SHA1 |
| EXP-EDH-DSS-DES-CBC-SHA | SSLv3 | DH(512) | DSS | DES(40) | SHA1 |
| EXP-ADH-DES-CBC-SHA | SSLv3 | DH(512) | None | DES(40) | SHA1 |
| EXP-ADH-RC4-MD5 | SSLv3 | DH(512) | None | RC4(40) | MD5 |

# Bibliography


[1] Apache Software Foundation, "SSLRequire directive." [Online]. Available: http://httpd.apache.org/docs/2.0/mod/mod_ssl.html#sslrequire

[2] Apache Software Foundation, "SSLCipherSuite Directive." [Online]. Available: http://httpd.apache.org/docs/2.0/mod/mod_ssl.html#sslciphersuite

[3] W. Stallings, *Cryptography and Network Security Principles and Practices*. Pearson Education International, 2003.

[4] E. W. Dijkstra, "On the role of scientific thought," in *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982, pp. 60–66.

[5] H. Zimmermann, "OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications [legacy, pre - 1988]*, vol. 28, no. 4, pp. 425–432, Apr 1980.

[6] K. Ostermann, "Reasoning about aspects with common sense," in *AOSD '08: Proceedings of the 7th international conference on Aspect-oriented software development*. New York, NY, USA: ACM, 2008, pp. 48–59.

[7] K. Beznosov, "Object security attributes: Enabling application-specific access control in middleware," in *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*. London, UK: Springer-Verlag, 2002, pp. 693–710.

[8] R. Venkatesan and S. Bhattacharya, "Threat-adaptive security policy," *Performance, Computing, and Communications Conference, 1997. IPCCC 1997., IEEE International*, pp. 525–531, Feb 1997.

[9] S. Alampalayam and A. Kumar, "An adaptive security model for mobile agents in wireless networks," *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. 3, pp. 1516–1521 vol.3, Dec. 2003.

[10] C. Chigan, L. Li, and Y. Ye, "Resource-aware self-adaptive security provisioning in mobile ad hoc networks," *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 4, pp. 2118–2124 Vol. 4, March 2005.

[11] P. Schneck and K. Schwan, "Dynamic authentication for high-performance networked applications," *Quality of Service, 1998. (IWQoS 98) 1998 Sixth International Workshop on*, pp. 127–136, May 1998.

[12] H. Hinton, C. Cowan, L. Delcambre, and S. Bowers, "SAM: Security Adaptation Manager," *Computer Security Applications Conference, 1999. (ACSAC '99) Proceedings. 15th Annual*, pp. 361–370, 1999.

[13] L. Marcus, "Local and global requirements in an adaptive security infrastructure," *International Workshop on Requirements for High Assurance Systems (RHAS 2003)*, Sept 2003.

[14] A. Shnitko, "Practical and theoretical issues on adaptive security," *Proceedings of FCS'04 Workshop on Foundations of Computer Security, Workshop on Logical Foundations of an Adaptive Security Infrastructure*, June 2004.

[15] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *J. Cryptology*, vol. 14, no. 4, pp. 255–293, 2001.

[16] A. K. Lenstra, "Key lengths," in *Handbook of Information Security*, H. Bidgoli, Ed. Wiley, 2006, pp. 617–635, volume 1.

[17] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee, and R. de Lemos, "08031 – software engineering for self-adaptive systems: A research road map," in *Software Engineering for Self-Adaptive Systems*, ser. Dagstuhl Seminar Proceedings, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., vol. 08031. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[18] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, July 2004.

[19] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "A taxonomy of compositional adaptation." [Online]. Available: http://citeseer.ist.psu.edu/mckinley04taxonomy.html

[20] S. M. Sadjadi, "A survey of adaptive middleware," 2003, Technical Report. MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan.

[21] B. Hashii, S. Malabarba, R. Pandey, and M. Bishop, "Supporting reconfigurable security policies for mobile programs," *Comput. Netw.*, vol. 33, no. 1-6, pp. 77–93, 2000.

[22] B. Timmerman, "A security model for dynamic adaptive traffic masking," in *NSPW '97: Proceedings of the 1997 workshop on new security paradigms.* New York, NY, USA: ACM, 1997, pp. 107–116.

[23] L. Teo, G.-J. Ahn, and Y. Zheng, "Dynamic and risk-aware network access management," in *SACMAT '03: Proceedings of the eighth ACM symposium on access control models and technologies.* New York, NY, USA: ACM, 2003, pp. 217–230.

[24] K.-D. Kang and S. H. Son, "Towards security and QoS optimization in real-time embedded systems," *SIGBED Rev.*, vol. 3, no. 1, pp. 29–34, 2006.

[25] M. El-Hennawy, Y. Dakroury, M. Kouta, and M. El-Gendy, "An adaptive security/performance encryption system," *Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference on Computer Engineering and Systems*, pp. 245–248, September 2004.

[26] T. Ryutov, C. Neuman, K. Dongho, and Z. Li, "Integrated access control and intrusion detection for web servers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 9, pp. 841–850, Sept. 2003.

[27] S.-W. Cheng, D. Garlan, and B. Schmerl, "Making Self-Adaptation an Engineering Reality," in *Proceedings of the Conference on Self-Star Properties in Complex Information Systems*, ser. LNCS, O. Babaoghu, M. Jelasity, A. Montroser, C. Fetzer, S. Leonardi, and A. van Moorsel, Eds., vol. 3460. Springer-Verlag, 2005.

[28] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, *Aspect-Oriented Programming.* Berlin, Heidelberg, and New York: Springer-Verlag, 1997, vol. 1241, pp. 220–242. [Online]. Available: http://citeseer.ist.psu.edu/kiczales97aspectoriented.html

[29] R. Hirschfeld and K. Kawamura, "Dynamic Service Adaptation," *International Conference on Distributed Computing Systems Workshops (ICDCSW)*, vol. 02, pp. 290–297, 2004.

[30] E. Truyen, B. N. Jrgensen, W. Joosen, and P. Verbaeten, "Aspects for run-time component integration," in *In Proceedings of the ECOOP 2000 Workshop on Aspects and Dimensions of Concerns, Sophia Antipolis and*, 2000.

[31] Z. Yang, B. H. C. Cheng, R. E. K. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley, "An aspect-oriented approach to dynamic adaptation," in *WOSS '02: Proceedings of the first workshop on Self-healing systems.* New York, NY, USA: ACM, 2002, pp. 85–92.

[32] P.-C. David, T. Ledoux, and N. M. N. Bouraqadi-Saâdani, "Two-step weaving with reflection using AspectJ," in *OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, 2001.

[33] C. Constantinides, T. Skotiniotis, and M. Störzer, "AOP considered harmful," in *European Interactive Workshop on Aspects in Software*, Berlin, Germany, September 2004, publikation. [Online]. Available: http://pp.info.uni-karlsruhe.de/uploads/publikationen/constantinides04eiwas.pdf

[34] M. Störzer and C. Koppen, "PCDiff: Attacking the Fragile Pointcut Problem, Abstract," in *European Interactive Workshop on Aspects in Software*, Berlin, Germany, September 2004, publikation. [Online]. Available: http://pp.info.uni-karlsruhe.de/uploads/publikationen/stoerzer04eiwas.pdf

[35] D. Bruce and N. Exon, "Alternatives to Aspect-Oriented Programming?" 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.6399

[36] A. Erradi, P. Maheshwari, and S. Padmanabhuni, "Towards a policy-driven framework for adaptive web services composition," in *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*. Washington, DC, USA: IEEE Computer Society, 2005, p. 33.

[37] R. Hao, L. Boloni, K. Jun, and D. C. Marinescu, "An Aspect-Oriented Approach to Distributed Object Security," in *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*. Washington, DC, USA: IEEE Computer Society, 1999, p. 23.

[38] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, and S. Zachariadis, "The RUNES middleware: A reconfigurable component-based approach to networked embedded systems," in *Proc. of 16th International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'05*. IEEE Press, 2005, p. 05.

[39] J. Knight, D. H. Alex, E. Wolf, A. Carzaniga, J. Hill, P. Devanbu, and M. Gertz, "The Willow Survivability Architecture," in *Proc. of the Fourth Information Survivability Workshop*, 2001.

[40] K. Scott and J. Davidson, "Software Security Using Software Dynamic Translation," 2001, Technical Report CS-2001-29, Department of Computer Science, Charlottesville, VA, USA.

[41] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K. E. Seamons, "Adaptive trust negotiation and access control," in *SACMAT '05: Proceedings of the tenth ACM symposium on access control models and technologies*. New York, NY, USA: ACM, 2005, pp. 139–146.

[42] M. Winslett, T. Yu, K. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu, "Negotiating trust in the web," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 30–37, Nov/Dec 2002.

[43] D. C. Schmidt, "Middleware for real-time and embedded systems," *Commun. ACM*, vol. 45, no. 6, pp. 43–48, 2002.

[44] D. C. Schmidt, D. F. Box, and T. Suda, "Adaptive: A dynamically assembled protocol transformation, integration and evaluation environment," *Concurrency - Practice and Experience*, vol. 5, no. 4, pp. 269–286, 1993.

[45] V. C. Zandy and B. P. Miller, "Reliable network connections," in *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking.* New York, NY, USA: ACM, 2002, pp. 95–106.

[46] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten, "Architecture and operation of an adaptable communication substrate," *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, pp. 46–55, May 2003.

[47] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design of the TAO real-time object request broker," *Computer Communications*, vol. 21, pp. 294–324, 1998.

[48] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, C. Magalhaes, and R. H. Campbell, "Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB," in *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000, pp. 121–143.

[49] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas, "An architecture for next generation middleware," in *Middleware, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'201998), Lake District.* UK, Springer-Verlag, pp. 15–18.

[50] J. A. Zinky, D. E. Bakken, and R. E. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems (TAPOS)*, vol. 3, no. 1, pp. 55–73, 1997.

[51] R. Baldoni, C. Marchetti, and A. Termini, "Active software replication through a three-tier approach," in *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02).* Washington, DC, USA: IEEE Computer Society, 2002, p. 109.

[52] J.-C. Fabre and T. Perennou, "A metaobject architecture for fault-tolerant distributed systems: the FRIENDS approach," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 78–95, Jan 1998.

[53] D. Sharp, "Reducing avionics software cost through component-based product line development." Salt Lake City, UT: The Software Technology Conference, April 1998.

[54] A. Singh, S. Gopisetty, L. Duyanovich, K. Voruganti, D. Pease, and L. Liu, "Security vs performance: Tradeoffs using a trust framework," *Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE / 13th NASA Goddard Conference on*, pp. 270–277, April 2005.

[55] A. Ganak and T. Corbi, "The dawning of the autonomic computing era," *IBM Systems Journal*, vol. 42, no. 1, pp. 5–18, 2003.

[56] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[57] Ö. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. P. A. van Moorsel, and M. van Steen, Eds., *Self-star Properties in Complex Information Systems, Conceptual and Practical Foundations*, ser. Lecture Notes in Computer Science, vol. 3460. Springer, 2005.

[58] M. Salehie and L. Tahvildari, "Autonomic computing: emerging trends and open problems," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[59] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, 2006.

[60] G. Dumont and M. Huzmezan, "Concepts, methods and techniques in adaptive control," *American Control Conference, 2002. Proceedings of the 2002*, vol. 2, pp. 1137–1150 vol.2, 2002.

[61] J. A. Mccann and M. C. Huebscher, "Evaluation issues in autonomic computing," 2004, pp. 597–608. [Online]. Available: http://www.springerlink.com/content/2q95n1w323nk28nc

[62] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," in *ULSSIS '08: Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems.* New York, NY, USA: ACM, 2008, pp. 23–26.

[63] "An Architectural Blueprint for Autonomic Computing," June 2006. [Online]. Available: http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

[64] A. Shnitko, "Adaptive security in complex information systems," *Proc. 7th Korea-Russia Int'l Symp. Science and Technology. IEEE Press.*, vol. 2, pp. 206–210, 2003.

[65] "Workshop on logical foundations of an adaptive security infrastructure," Turku, Finland source = http://www.aero.org/wolfasi,, July 2004.

[66] C. Montangero and L. Semini, "Formalizing an adaptive security infrastructure in $mob_{adtl}$," *Workshop on logical foundations of an adaptive security infrastructure*, 2004.

[67] C. J. Lamprecht and A. van Moorsel, "Adaptive ssl design, implementation and overhead analysis," in *First International Conference on Self-Adaptive and Self-Organizing Systems*, J. Martin-Flatin and M. J. et al. (eds.), Eds. Cambridge, Massachusetts: IEEE Computer Society, July 2007, pp. 289–292, submission for Applications Track.

[68] S. H. Son, R. Zimmerman, and J. Hansson, "An adaptable security manager for real-time transactions," in *In Euromicro Conference on Real-Time Systems*, 2000, pp. 63–70.

[69] T. Dierks and E. Rescorla, "RFC4346: The Transport Layer Security (TLS) protocol version 1.1," United States, April 2006, IETF Network Working Group.

[70] W. Stallings, *Cryptography and Network Security: Principles And Practices*. Prentice Hall, 2008.

[71] W. Mao, *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference, 2003.

[72] C. Adams and S. Lloyd, *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[73] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[74] R. Rivest and B. Kaliski, "RSA problem," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed. Springer, 2005.

[75] H.-M. Sun and M.-E. Wu, "An approach towards rebalanced RSA-CRT with short public exponent," Cryptology ePrint Archive, Report 2005/053, 2005, http://eprint.iacr.org/.

[76] J. Jonsson and B. Kaliski, "Public-key cryptography standards (PKCS) #1: RSA cryptography specifications version 2.1," United States, 2003, IETF Network Working Group.

[77] W. Freeman and E. Miller, "An experimental analysis of cryptographic overhead in performance-critical systems," in *MASCOTS '99: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. Washington, DC, USA: IEEE Computer Society, 1999, p. 348.

[78] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," *Notices of the American Mathematical Society (AMS)*, vol. 46, no. 2, pp. 203–213, 1999. [Online]. Available: http://citeseer.ist.psu.edu/285207.html

[79] D. Eastlake, "RSA/SHA-1 SIGs and RSA keys in the domain name system (DNS)," United States, 2001, IETF Network Working Group.

[80] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *EUROCRYPT*, 1994, pp. 92–111.

[81] M. Wiener, "Cryptanalysis of short RSA secret exponents," *Information Theory, IEEE Transactions on*, vol. 36, no. 3, pp. 553–558, May 1990.

[82] D. Boneh and G. Durfee, "Cryptanalysis of RSA with private key $d$ less than $n^{0.292}$," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1339–1349, Jul 2000.

[83] D. Boneh and H. Shacham, "Fast variants of RSA," *RSA Cryptobytes*, vol. 5, no. 1, pp. 1–9, Winter/Spring 2002.

[84] "Java cryptography extensions," Sun Microsystems, Inc, JavaTM Cryptography Extension, 2005. [Online]. Available: http://java.sun.com/products/archive/jce/

[85] "Java Cryptix Libraries," The Cryptix Foundation Limited, 2005. [Online]. Available: http://www.cryptix.org/

[86] OpenSSL. [Online]. Available: http://www.openssl.org/

[87] Apache Software Foundation, "Trusted Services Integration Kit (TSIK)." [Online]. Available: http://incubator.apache.org/projects/tsik.html

[88] P. Tomlinson, "An analysis of real time security in an online trading stocks and shares system," Master's thesis, Newcastle University, Computing Science, 2000.

[89] C. Lamprecht and A. van Moorsel, "Performance measurement of web services security software," N. Thomas, Ed. University of Newcastle upon Tyne, School of Computing Science: 21st UK Performance Engineering Workshop, July 2005, pp. 11–20.

[90] C. Lamprecht, A. van Moorsel, P. Tomlinson, and N. Thomas, "Investigating the efficiency of cryptographic algorithms in online transactions," *International Journal of Simulation: Systems, Science & Technology*, vol. 7, no. 2, pp. 63–75, 2006.

[91] B. Schneier, *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995.

[92] K. Schmeh, *Cryptography and Public Key Infrastructure on the Internet*. Wiley, 2001.

[93] D. R. Stinson, *Cryptography: Theory and Practice*, 3rd ed., ser. Dicsrete Mathematics and its Applications, K. H. Rosen, Ed. Chapman and Hall, 2006.

[94] A. K. Lenstra, "Further progress in hashing cryptanalysis," Feb 2005. [Online]. Available: http://cm.bell-labs.com/who/akl/hash.pdf

[95] OpenSSL Linux speed tester. [Online]. Available: http://www.openssl.org/docs/apps/speed.html

[96] Apache Software Foundation. [Online]. Available: http://www.apache.org/

[97] T. Imamura, B. Dillawa, and E. Simon, "W3C Recommendation, XML Encryption Syntax and Processing," Dec 2002. [Online]. Available: http://www.w3.org/TR/xmlenc-core

[98] J. Aslam, S. Rafique, and S. Tauseef-ur-Rehman, "Analysis of real-time transport protocol security," *Information Technology Journal*, vol. 3, no. 3, pp. 311–314, 2004.

[99] B. Kaliski and J. Staddon, "PKCS ♯1: RSA cryptography specifications version 2.0," United States, Oct 1998, IETF Network Working Group.

[100] "The Legion of the Bouncy Castle RSA libraries v1.30." [Online]. Available: http://www.bouncycastle.org

[101] B. Laurie and P. Laurie, *Apache: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002.

[102] R. S. Engeschall and B. Laurie, "mod_ssl." [Online]. Available: http://httpd.apache.org/docs/2.0/mod/mod_ssl.html

[103] Apache Software Foundation, "Multi-Processing Modules." [Online]. Available: http://httpd.apache.org/docs/2.0/mpm.html#introduction

[104] D. Mosberger and T. Jin, "HTTPerf - A Tool for Measuring Web Server Performance," in *First Workshop on Internet Server Performance*. ACM, 1998, pp. 59–67.

[105] J. T. J. Midgley, "Autobench tool." [Online]. Available: http://www.xenoclast.org/autobench/

[106] M. Arlitt, "Characterizing web user sessions," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 2, pp. 50–63, 2000.

[107] Jakarta Commons HttpClient. [Online]. Available: http://hc.apache.org/httpclient-3.x/index.html

[108] S. Godard, "iostat." [Online]. Available: http://pagesperso-orange.fr/sebastien.godard/

[109] B. R. Haverkort, "Can we quantitatively assess security?" in *Workshop on Empirical Evaluation of Dependability and Security (WEEDS 2006)*. IEEE, June 2006.

[110] C. Griffin, B. Madan, and K. Trivedi, "State space approach to security quantification," in *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 83–88.

[111] Apache Software Foundation, "SSL variables." [Online]. Available: http://httpd.apache.org/docs/2.0/mod/mod_ssl.html#table3