Secure Communications

The Masters Series

Jericho in depth…

Secure Communications

End-to-end encryption in Jericho networks

Alina Stan

Capgemini's Security & Innovation Research Centre, based in the Netherlands, focuses on near future IT Security solutions. The Jericho forum's vision on network de-perimeterization and Boundaryless Information flow™ has been the starting point for this research centre. Research papers from this centre appear in two distinct categories;

1. The Master Series

Researcher holding a masters degree in Informatics or are in the process of obtaining a master degree publish in the Master Series. The participating University and the Capgemini Security & Innovation Research centre have approved publications in this category.

Publications in this Series for 2008;

- Jericho in depth… Secure Communications by A. Stan
- Jericho in depth… The road to Jericho by A. Stan

Planned publications in this Series for 2008;

- Demystifying trust by F. van Leijden
- Jericho in depth… Automated Security Classification by K. Clark
- Jericho in depth… Trust Management for Trust brokers by A. Demarteau

2. The Bachelor Series

Researchers holding a bachelors degree in Informatics or in the process of obtaining a bachelors degree publish in the bachelor series. Their University and the Capgemini Security & Innovation Research centre have approved publications in this category.

Publication in this series for 2008;

- Jericho in depth… Endpoint security by L. Teheux
- Jericho in depth… Authentication and Accounting by E. Barannikov
- Jericho in depth… Trust broker Services by A. Bruning
- Jericho in depth… Trust broker framework by A. Bruning

Planned publications in this series for 2008;

- Jericho in depth… Controlling the COA framework by J. Willemsen
- Jericho in depth… Fully ASP based by D. Hanenberg & F. Aardoom

Preface

Since the dawn of the Internet at the ending of 1969 a lot has changed, I'm sure nobody will disagree with a statement like that. During the last couple of years however, we seem to have hit a mid-life crisis of the Internet. The sudden boost of Internet technology over the past decade does not fit well with our outdated design principles for network security. Most organizations hold tight to their fortress approach in trying to protect the internal network from the hostile Internet. Understandable, but not really realistic. In the Netherlands, we are particularly proud of our water management techniques. In a country that lays for more then sixty percent below sea level we know that we have to build and maintain solid dikes to prevent our country from flooding. Having holes in these dikes quickly diminishes the whole purpose of have a dike. The same holds true for perimeter defence in computer networks. Information leakage via email, hyves, my space or mobile data solutions like iPod or USB diminishes the purpose of perimeter security. Today's business world is one of collaboration, one of working together., one of global markets. The Internet is the ideal candidate to support this collaboration. The Jericho Forum (Open Group), formed by Security professionals from the largest organisations in the world described their vision of network de-perimeterization and boundryless Information flow™ in various publications. These visions formed the starting point for Capgemini's Security & Innovation Research Centre.

Together with the best universities in the Netherlands, Capgemini's offers academic researchers and graduate students to ability to conduct empirical academic research into the topic of Collaboration Oriented Architectures or to conduct feasibility studies into the Jericho Forums visions.

Marco Plas

Head of Jericho Research
Capgemini Security & Innovation Research Centre
Capgemini Netherlands

Contents

1. Introduction

Nowadays, there are new ways of doing and conducting businesses, different from the traditional business models. Simply defined, a business model is the method of doing business by which a company generates revenue[1]. The online collaboration and electronic commerce among multiple business actors are increasing and this involves sharing of sensitive data over Internet. Sensitive data might include credentials used for authentication, or data such as credit card numbers, or bank transaction details, patient healthcare information. Defining what sensitive data is depends on the security policy of each organization and on the security levels attached to data (e.g. confidential, secret, top secret). The Computer Security Act 2 of 1987 provided a broad information definition for the term sensitive information: "or modification of which could adversely affect the national interest or the conduct of federal programs, or the privacy to which individuals are entitled under section 552a of title 5, United States Code (the Privacy Act), but which has not been specifically authorized under criteria established by an Executive Order or an Act of Congress to be kept secret in the interest of national defense or foreign policy." Consequently, protecting sensitive information means providing at least the following security services: confidentiality, integrity and availability of information. The Internet business models continue to evolve and, given the rapid advances and expansive growth of information and communication technology, new and interesting variations will arise in the future[3, 4] (Turban et al., 2006).

The new business models (B2B, B2C, B2E, G2G, electronic markets, e-shops, auctions etc.) that appeared demand for new approaches of providing security. These are based on electronic, mobile transactions and users. Therefore, a new security architecture where each device is capable to protect itself is needed.[5] In the Internet age, the mission of securing network communications effectively is vital for organizations and individuals as well (Mar-Elie et al., 2005; Ramachandran, 2002; Dam et al., 1996).

Jericho Forum suggests that the perimeter approach for providing security is not suitable anymore and does not meet the increasing demands for security and user mobility that are inherent of the e-business models. The old ways for providing protection with an organizational security boundary are not adequate anymore and become obsolete. The traditional security mechanisms do not meet anymore the demands of businesses over an open, flexible and Internet-driven enterprise environment. This new security approach that sustains collaboration and commerce over open networks, within and between organizations, is based on security architecture and design approach entitled 'de-perimeterization'. The new security model, its principles (commandments) and a series of white papers are proposed and promoted by Jericho Forum.

---

[1] Michael Rappa http://digitalenterprise.org/models/models.html accessed March 2007
[2] http://csrc.nist.gov/publications/nistbul/csl92-11.txt accessed June 2007
[3] Idem 1
[4] Jericho Forum http://www.opengroup.org/projects/jericho/index.tpl White Paper Business Case for De-perimeterization (January 2007)
[5] Jericho Forum http://www.opengroup.org/projects/jericho/index.tpl

In this book we will focus on proposing and describing end-to-end encryption solutions for secure communications within the context of Jericho Project. For providing end-to-end encryption, cryptography algorithms and security protocols are considered. Cryptography enables two or more parties to communicate and exchange information securely over insecure channels. According to Ramachandran (2002) "the success of the Internet as a marketplace for services and information depends on the strength of our cryptographic protocols and algorithms.

Jericho Forum. Jericho Project

Jericho Forum[1] is an international community, composed mainly of IT organizations, dedicated to the development of open standards to enable secure and de-perimeterised information flows across networks. The members of Jericho Forum recognize that the current security mechanisms that protect business information will not scale in the near future to meet the requirements for protecting the increasing volumes of transactions and data in a continuously extending collaborative business environment. Jericho Forum envisions a shift in the security world from the traditional network perimeter down to the individual networked computers and devices – and ultimately to the level of the data itself. Thus, the security perimeters will disappear step by step. This process has been described as 're-perimeterization', followed by ultimate 'de-perimeterization'. This forum explores the potential to develop security architectures that support de-perimeterized business-to-business networking. The need for such standards has been growing over the past years as organizations are conducting more and more businesses over the Internet. So, the challenge to remove or to move away from the security perimeter has to be tackled. This does not necessarily mean that the firewalls that provide basic network protection will be removed, but apart from these, the individual systems and the data need to have the capability of protecting themselves. De-perimeterization refers to redesigning the security perimeters in order to foster collaboration between and within organisations over open networks. Jericho Forum sustains this principle and offers standards, design principles, named commandments[2], and guidance in order to help the creation and the broad adoption of such security technology. According to Jericho Forum[3], a solution for a de-perimeterized network requires that every component is independently secure, demanding systems and data protection on multiple levels. In order to design and build a de-perimeterized solution, it will be needed a combination of at least the following elements: encryption, inherently-secure computer protocols, inherently-secure computer systems, data-level authentication.

Jericho Forum aims to stimulate a market that provides solutions for de-perimeterized networks. These solutions should use open standards, improve interoperability and integration, both within the IT systems and among the different businesses.

Jericho Project has started in December 2006 at Capgemini Nederland B.V. under the initiative of Drs. Marco Plas, in the department Telecom, Travel and Utilities. Initially, the

[1] Jericho Forum http://www.opengroup.org/projects/jericho/index.tpl
[2] Jericho Forum Commandments http://www.opengroup.org/jericho/commandments_v1.1.pdf accessed April 2007
[3] Jericho Forum http://www.opengroup.org/jericho/ accessed April 2007

project team comprised five students that conduct research for defining a solution for Jericho networks. The Jericho Project aims at providing a solution based on the commandments proposed by Jericho Forum. The solution is defined by a combination of the following research subjects: authentication, authorization, accounting, trust broker, endpoint security, data classification and end-to-end encryption. More details about each research topic and about their importance within Jericho Project will be provided in Chapter two.

End-to-end encryption

The need for information security is fostered by the broad use of Internet and emerging collaborative business models. Sensitive information is found nowadays in three states: in storage, in transit and in the process of transformation from storage to transit. All kinds of information are classified as sensitive: business communication, e-mails, electronic transfers and other financial transactions, technology and trade secrets, personal records containing personal information etc. Basically, sensitive information refers to electronic information or data records that, if used improperly, can harm the information subjects, information owners or information users or may contravene other public-policy interests.[1] The data itself can be found in three states: in storage, in process of transformation (e.g. data at rest from storage is being encrypted), and in transit. Data has to be protected in all these three states. In the context provided by Jericho Project we focus in this book on exploring and proposing valid solutions for protecting data in transit, namely for end-to-end encryption. The aim is to ensure the integrity and confidentiality of the data. In order to do this, the entities need to communicate securely, and for this the transferred data has to be encrypted.

Research objective and research questions

The aim of this book is to investigate and make recommendations regarding the most adequate use of cryptography for security protocols that can be used for providing end-to-end encryption in Jericho networks. The possibilities offered by cryptography for implementing and deploying suitable security protocols within Jericho networks are further researched within this book.

The following questions will be dealt with and their responses will be analyzed further in this book:

- What are the requirements for secure communications in Jericho networks?
- What is the range of solutions that can be used for end-to-end encryption?
- Which is/are the recommended end-to-end encryption solution(s) in the context of Jericho Project?
- Which are the most adequate cryptographic algorithms, in terms of security offered and performance, to be used for security protocols that offer end-to-end encryption?

Research methods

[1] http://dig.csail.mit.edu/2006/tami-portia-accountability-ws/summary#Overview accessed May 2007

In order to come with valid recommendations for the research questions mentioned above, we will perform a thorough research of the cryptographic algorithms and their use and influence on the performance of security protocols for secure communications in Jericho networks. In the pursuit of this research, firstly, we will define the conceptual model for Jericho networks. Secondly, a theoretical foundation of cryptography and security protocols offers the basis for performing a comparison of the different algorithms and protocols in terms of security, performance, key size. Based on the results of the comparisons, different case scenarios of security protocols that can be used for obtaining end-to-end encryption in Jericho networks will be analyzed. Further aspects regarding the implementation and deployment of the security protocols will be researched on the vendor market.

Scope

The purpose of this book is to investigate different solutions for end-to-end encryption and to propose the most suitable solutions that meet the requirements of protecting data in transit in Jericho networks. The book aims to provide new approaches and recommendations for securing the sensitive data that is transferred across Internet. Within the scope of this research, firstly, the theoretical foundation of the security protocols and cryptographic algorithms will be discussed. Then, their implications and applications in the context of Jericho networks will be investigated. Moreover, we will conduct research in detail regarding the security provided at different levels in OSI model (e.g. network, transport, application level) and perform a thorough description and an objective comparison of open standards available such as IPsec, SSL/TLS. However, the focus of the book will be to study end-to-end encryption. For the general purpose of this book we need to explore also the security features offered by means of cryptography. In this respect, we will investigate, describe and compare the most relevant cryptographic algorithms. The cryptographic algorithms will be compared in terms of complexity, performance and key size. Besides this, their impact on the performance and security features of the solutions for end-to-end encryption will be analyzed. Furthermore, we will give recommendations about which are the most adequate cryptographic algorithms to be used for the analyzed security protocols. Finally, it is worth mentioning here that the end goal of Jericho Project is to offer possible solutions for Jericho networks, and to implement and test a prototype at the Labs of Capgemini Nederland B.V.

So, we conclude that initially, the general purpose of Jericho Project is to provide a roadmap for further implementing and testing a prototype based on the commandments proposed by Jericho Forum.

Book outline

Including this introductory chapter, the book is structured in 5 chapters presenting our research and findings about different aspects of cryptography and its use for designing security protocols that provide adequate security services within Jericho Project. Further on, we briefly review the contents of these chapters. Chapter 2 introduces the business case for Jericho Project within Capgemini Nederland B.V. Chapter 1 and 2 provide mainly the research and business context for the purpose of this book. Chapter 3 covers the topic of security protocols that provide end-to-end encryption. Firstly, we will

investigate the security features offered by Secure Sockets Layer / Transport Layer Security (SSL/TLS). SSL/TLS is used for securing electronic commerce and communications on Internet, and is considered by professionals an elegant and efficient protocol (Stamp, 2006, p.5). Then, we will explore Internet Protocol Security (IPsec), a security protocol that is more complex than SSL, but offers some similar security services. Chapter 4 presents a suite of cryptographic algorithms and is intended to be a guide to cryptography. In this chapter, the following topics are presented: asymmetric key cryptography, symmetric key cryptography, block ciphers and their operation modes, stream ciphers, message authentication codes, hash functions, cryptography in the real world. Also, in this chapter we will provide a comparison of the cryptographic primitives in terms of security offered and performance in order to establish which are the most adequate to be used for providing end-to-end encryption in Jericho project. Chapter 5 summarizes the conclusions of the book and provides possible directions for future research in the context of Jericho Project.

2. Scenario: Jericho Project

In this chapter we provide an overview of Jericho Forum. Following, we present the research topics within Jericho Project initiated at Capgemini Nederland B.V.

Jericho Forum

Jericho Forum[1] is an international IT security thought-leadership group dedicated to defining ways to deliver effective IT security solutions that will match the increasing business demands for secure IT operations over Internet. Consequently, Jericho Forum aims to drive and influence development of security standards that will meet future business needs. These standards are intended to facilitate the secure interoperation, collaboration and commerce over Internet, and to facilitate the implementation and deployment of a new security architecture and design approach based on the principle of "de-perimeterization". The members of Jericho Forum recognize that the current security mechanisms that protect business information will not scale in the near future to meet the requirements for protecting the increasing volumes of transactions and data in a continuously extending collaborative business environment. The Forum introduces the concept of "de-perimeterization" and encourages organizations to look at securing the data itself rather than the infrastructure that supports it. Jericho Forum envisions a shift in the security world from the traditional network perimeter down to the individual networked computers and devices – and ultimately to the level of the data itself. This process has been described as "re-perimeterization", or "de-perimeterization". This Forum explores the possibility to develop common security architectures to support de-perimeterized business-to-business networking. The need for such standards has been growing over the past years as organizations are conducting more and more businesses over the Internet. So, the challenge to remove or to move away from the security perimeter has to be tackled. This does not necessarily mean that the boundary firewalls that provide basic network protection will be removed, but apart from these, the individual systems and the data need to have the capability of protecting themselves. Stamp et al. (2005) suggested in a Forrester Analyst Report, a roadmap that Jericho Forum envisions and aims to materialize in the coming years. The authors proposed a four stage roadmap for achieving the goals of Jericho Forum:

- *Make services available across the perimeter:* Organizations are already making their services available across the Internet using technologies like Web Services, and security protocol such as SSL/TLS, XML encryption.
- *Next, remove the perimeter altogether:* The next stage is to reduce the importance of the network boundary (firewalls and intrusion prevention systems) as a security control. Traditionally, the perimeter firewall becomes one of a series of devices to block malicious traffic. But in a de-perimeterized network, the focus is on authenticating entities and giving them the adequate access level.
- *Develop a standards-based approach to data access:* Once the perimeter fades away, an open, standardized way for entities' authentication and

[1] http://www.opengroup.org/jericho/

authorization has to be decided upon. Open and inherently secure standards should be made designed and used in Jericho networks.

Jericho Forum aims to stimulate a market that provides solutions for the de-perimeterization challenge. These solutions should use open standards, improve interoperability and integration, both within the IT systems and among the different businesses.

- *Then, control access to the data, not the underlying infrastructure:* Finally, organizations will implement a security model that guarantees data confidentiality and integrity independent of the status of the data (in storage, in processing, in transit). Organizations will only transfer data between authenticated and authorized parties, and the information regarding the encryption capabilities are sent along with the data itself.

Consequently, Jericho Forum explores the potential to develop security architectures to support de-perimeterized business-to-business networking. De-perimeterization refers to redesigning the security perimeters in order to foster collaboration between and within the organisations over open networks. Jericho Forum sustains this principle and offers standards and guidance in order to help the creation and, at the same time, adoption of such security technology. Actually, de-perimeterization is a concept that describes how to meet the business needs for the businesses without a hardened perimeter. Moreover, this business-driven security solution provides defence in depth. The members of Jericho Forum have written and published a series of general white papers and position papers[1] about topics of interest for designing, implementing and deploying a network architecture based on the principle of de-perimeterization.

Based on the papers published by Jericho Forum, and especially on the paper presenting the Jericho Forum Commandments, the following requirements and features are deducted for Jericho networks:

- Security mechanisms must be pervasive and scalable
- All devices must be capable of maintaining their security policy on untrusted networks
- All people, processes and technologies must have been authenticated and transparent levels of trust are necessary for any transaction to take place
- Mutual trust assurance levels must be determinable; in fact, de-perimeterization requires a universal trust infrastructure
- Authentication, authorization and accounting must interoperate with other implementations outside an organization's area of control
- Access to data should be controlled by security attributes of the data itself

Further, these requirements will be investigated in the research topics formulated within Jericho Project at Capgemini Nederland B.V.

Jericho Project

For Jericho Project, the security strategy is built on defining and implementing new approaches for authentication, authorization, accounting, endpoint security, data

[1] http://www.opengroup.org/jericho/publications.htm

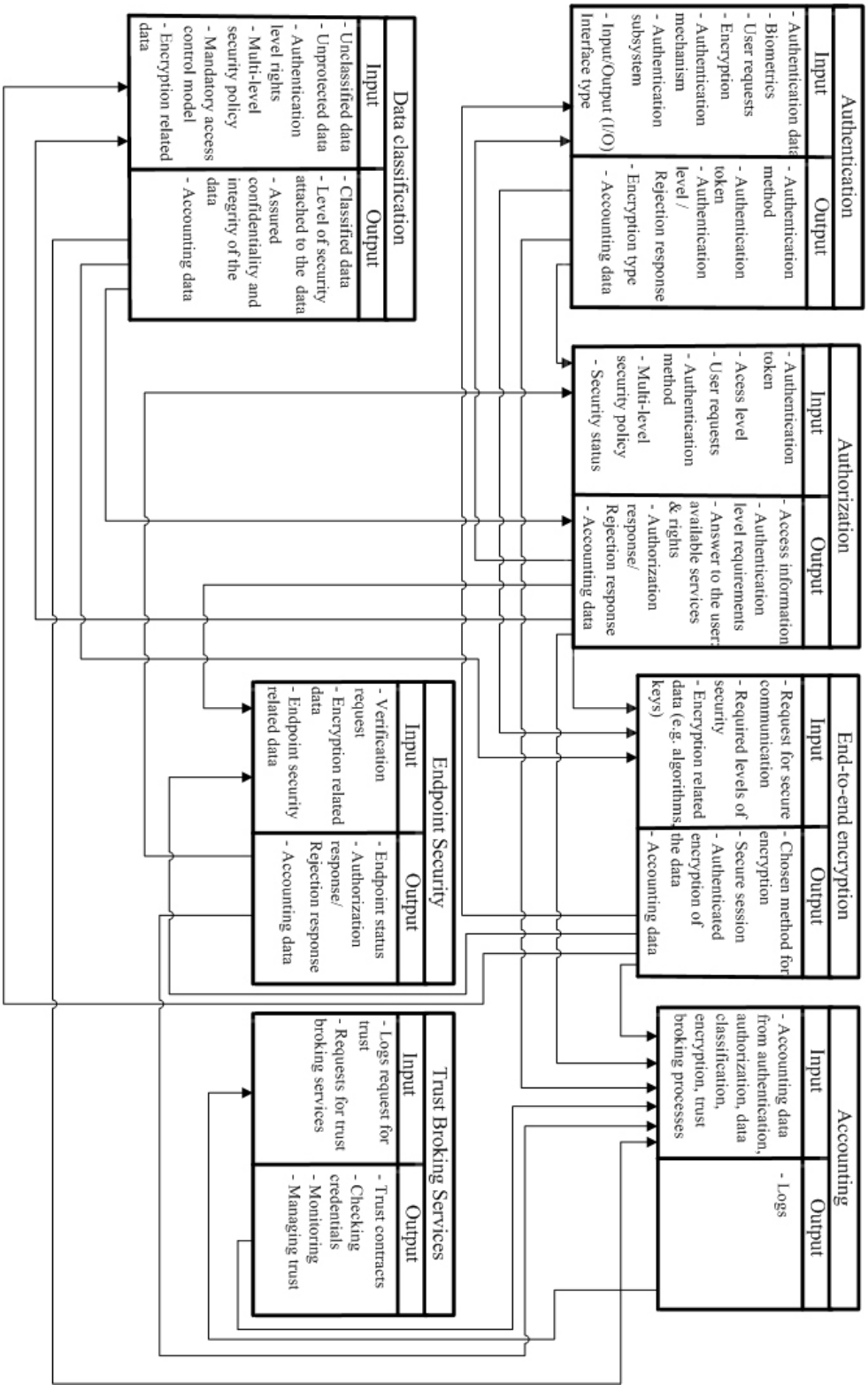
classification & information leakage, and secure communications in order to provide security services such as privacy, authentication of the entities, non-repudiation, integrity in de-perimeterized networks. These topics will be addressed in detail in the research conducted for Jericho Project. Capgemini, by being one of the members of Jericho Forum, is actively involved in providing and designing a new security architecture based on the principles and commandments emitted by Jericho Forum. There is a direct interest within the company to provide a sound solution for a new security architecture based on Jericho Forum Commandments[1]. Within Capgemini, drs. Marco Plas initiated this project and formed a research group, in order to come with a tangible solution for Jericho networks. The goal of Jericho Project is to develop a new security architecture and design approach that will enable business to grow safely and securely in an open, Internet-driven, networked world. In this new security architecture, each device is capable of protecting itself and each asset of the network is individually protected. De-perimeterization requires security to be at the heart of the organisation's distributed technology architecture. Security has to be implemented in end-user devices, application services, and it has to efficiently and effectively protect organisations' critical information assets themselves. The aim of Jericho Project is to define a roadmap for the implementation of this new defined security architecture. The research group for Jericho Project is composed of five students under the direct supervision of Drs. Marco Plas. The work within the research group is directed towards a comparative research of possible methods, models, technologies, cryptographic algorithms, security protocols from the network security area. The goal of the research within Jericho Project is to provide possible solutions for the new security architecture in de-perimeterized networks.
The research conducted for Jericho Project is divided into five distinct parts that focus on the following inter-connected topics of research:

- AAA Framework
  - Authentication
  - Authorization
  - Accounting
- Trust Broker
- Endpoint security
- Data classification
- End-to end encryption

The interconnections between these topics of research are illustrated in Figure 1.

Figure 1: Process Schema for the research modules in Jericho Forum Project

[1] www.opengroup.org/jericho/commandments_v1.1.pdf accessed May 2007


Authentication
Input
- Authentication data
- Biometrics
- User requests
- Encryption
- Authentication mechanism
- Authentication subsystem
- Input/Output (I/O) Interface type
Output
- Authentication method
- Authentication token
- Authentication level / Rejection response
- Encryption type
- Accounting data
Authorization
Input
- Authentication token
- Acess level
- User requests
- Authentication method
- Multi-level security policy
- Security status
Output
- Access information
- Authentication level requirements
- Answer to the user: available services & rights
- Authorization response/ Rejection response
- Accounting data
End-to-end encryption
Input
- Request for secure communication
- Required levels of security
- Encryption related data (e.g. algorithms, keys)
Output
- Chosen method for encryption
- Secure session
- Authenticated encryption of the data
- Accounting data
Accounting
Input
- Accounting data from authentication, authorization, data classification, encryption, trust broking processes
Output
- Logs
Data classification
Input
- Unclassified data
- Unprotected data
- Authentication level rights
- Multi-level security policy
- Mandatory access control model
- Encryption related data
Output
- Classified data
- Level of security attached to the data
- Assured confidentiality and integrity of the data
- Accounting data
Endpoint Security
Input
- Verification request
- Encryption related data
- Endpoint security related data
Output
- Endpoint status
- Authorization response/ Rejection response
- Accounting data
Trust Broking Services
Input
- Logs request for trust
- Requests for trust broking services
Output
- Trust contracts
- Checking credentials
- Monitoring
- Managing trust

AAA Framework

Authentication[1]

According to 8th Commandment of Jericho Forum[2] "Authentication, authorization and accountability must interoperate /exchange outside of your locus / area of control", identity data must be not usable only within one domain, but also be inter-exchangeable among multiple parties.

Authentication is the process which establishes a subject's identity. It plays a very important role in Jericho Project. Data transaction between entities is allowed to happen only after the subject's successful authentication. Authentication can use a single or multiple factors. The more factors are present the more assurance one receives of the person's identity. The aim of the research is to identify the methods and technologies with which a user can be authenticated in federated and user-centric identity systems. The scope of the research is limited to the user – trust broker interaction. Some topics of trust and identity systems will be covered by the research in order to determine which data may be needed by the trust broker. Authentication is interconnected with other parts of the research. Authentication is an essential part of the AAA framework. Authentication delivers identity data, which finds its further usage in the authorisation process. Accounting keeps logs of all the authentication process for the further auditing or investigation. Some identity data may also be used by the encryption process to encrypt either authentication or data transaction.

The deliverables of the Authentication research within Jericho Forum Project will be:

- The logical requirements for authentication in Jericho Forum network
- Comparison of the identity systems
- Comparison of the identity models
- Determine which technical solutions may be used in implementation of the Jericho forum network
- A practical recommendation concerning the suitable solution

Authorization

As described within the Federated Identity Position Paper[3] (2004), after establishing identities, it should be determined what rights are applicable to entities' requests. Within the AAA framework, this process is known as authorization. At the moment, authorization is dependent on the authentication process. Only after an entity has established its credentials, the authorization process determines what rights are associated with the resource it attempts to access and, acting upon this, allows or denies the request. As such, Authorization research topic is an essential part of the Jericho Project. Without authorization, no controlled network interactions can exist. A new architecture that can add flexibility and inter-area operability to this concept is needed. The claims-based architecture is a promising example. The claims-based architecture is a promising example. Whereas until now users had to provide proof of identity before the authorization process could commence, authentication and authorization can be combined to provide a flexible and more effective solution. In the

[1] Evgeny Barannikov, Jericho in depth… Authentication & Accounting, Capgemini, 2008

[2] www.opengroup.org/jericho/commandments_v1.1.pdf accessed May 2007

[3] https://www.opengroup.org/jericho/publications.htm accessed May 2007

context of Jericho Project, the research about authorization has several interactions with other research topics. The Authentication process needs to authenticate entities before authorization can take place, whilst the accounting process needs to gather data and process it. In addition, the Endpoint security process may need to deliver information that can be used to determine authorization rights.

The scope of the research is limited by the trust broker and authentication process. The goal of this research is the establishment of a functional model. In order to do this, the following steps have to be researched:

- Determine the role of Authorization within the Jericho Forum network
- Determine logical requirements for Authorization process
- Determine interaction requirements with other processes within this model
- Determine technical requirements of the Authorization process
- Compare the established requirements with currently available solutions
- Recommend currently available solutions

Possible results of this research on Authorization process include the creation of a new, universal standard which can be applied to Authorization, the establishment of a communication channel with Authorization solution companies in order to Jericho-enable future versions of their products and the description or actual implementation of an available Authorization solution in a prototype environment.

Accounting[1]

"Authentication, authorisation and accountability must interoperate /exchange outside of your locus / area of control" is stated in Jericho Forum Commandment number 8[2]. Auditing is a process that collects and processes the log data which is delivered by other processes. Separate IT systems in the enterprise architecture provide log data, which is processed independently from each other. In case of security breach or performance analyses multiple logs must be accessed and analysed. Auditing policy must comply with the legislation of the country, where the company is established. Also log retention period is dictated by the security policy of the corporation and the laws of the country. Auditing process may deliver data to other processes that can determine through certain algorithms the trustworthiness of the authenticated party. This data may also be accessed by the third party that objectively and independently establishes the entity's reputation. Anonymity is a very important factor here. Not all data may be disclosed to other parties. The goal of this sub-project is to research and define technology which could consolidate the logs from the multiple systems and provide certain log data to other processes. The scope of the research is limited by the trust broker and network services. Accounting research topic is connected to every single process within the Jericho Forum networks. Auditing policy is dictated by the accounting process and flows to other processes. Information from other modules is collected and processed in the single log repository.

The following deliverables of the Accounting research within Jericho Forum Project will be produced:

[1] Evgeny Barannikov, Jericho in depth… Authentication & Accounting, Capgemini, 2008
[2] www.opengroup.org/jericho/commandments_v1.1.pdf accessed May 2007

- The logical requirements for the Jericho Forum network
- Research and analysis of the existing log consolidation software
- Comparison chart of the software
- A practical recommendation concerning the suitable solution

Trust Broker[1]

Below it is described how the Trust Broker research topic is tackled within Jericho Project. The main purpose of the Trust Broker is that it will act in trust. In our vision it can do this as a neutral third party that will facilitate certain services from which it can not take any advantages, except some compensation from the two or the several other parties involved. These services can take place on all kinds of areas because the service, or data which originate from the service, is frequently carried out at one of the acting parties. The only thing what the Trust Broker will supervise is that if these parties can trust each other, if these services are carried out effectively, if it is even possible to carry out these services, and further if they are performed adequately. This means that a Trust Broker will act between two or several parties who want to able to do business with each other, but need an extra factor of faith to do this. For this, the Trust Broker must determine if every party can be trusted and if it is still the same party when the agreement was concluded. Therefore, in principle, the Trust Broker creates a Circle of trust between these two or several parties.

Worldwide computer crime has increased enormous. And the crime that occurs the most has always something to do with unauthorized people accessing certain sensitive data by which they can perform several illegal activities, such as transferring money, hold people for ransom, sell corporate secrets or sell peoples (digital) identities. To resolve or at least to reduce these problems users have to get control back over their identities and must be able decide independently what is right and wrong on Internet. In order to accomplish this, it is necessary that all people can perform some kind of simple check if they are communicating with the right party. In order to do this there must come some kind of simple and unambiguous trust system. In short, to reduce the crimes in the digital realm it is necessary to have a system that can measure or define someone's trust level, and this will be the main goal of this research topic within Jericho Project. The main goal of this research part of Jericho Project is to define the actions of a Trust Broker within the context of a Jericho network. As the meaning of the concept "Trust Broker" suggests, its main function will be to act in trust. The purpose of the trust Broker research is to find out how a Trust Broker would function within a federated and a user-centric network. We will focus on federated network because of its higher acceptance level.

For this research topic within Jericho Project, there will be used the following Jericho Forum commandments:

- Jericho Forum Commandment (JFC) 2: Security mechanisms must be pervasive, simple, scalable & easy to manage.

[1] Adriaan Bruning, Trust Broker in Jericho Project, Capgemini, 2007

- JFC 4: Devices and applications must communicate using open, secure protocols.
- JFC 5: All devices must be capable of maintaining their security policy on an untrusted network.

And particularly:

- JFC 6: All people, processes, technology must have declared and transparent level of trust for any transaction to take place.
- JFC 7: Mutual trust assurance levels must be determinable.
- JFC 8: Authentication, authorization and accountability must interoperate / exchange outside of your locus / area of control.

With these Jericho Forum commandments, trust Broker research topic is connected with the AAA-framework (C8), encryption (C4), endpoint security (C5,C4) and in order to facilitate services there must be a connection with data classification (C9,C10,C11). Likely the outcome of the research about Trust Broker will be largely influenced by the manner of authentication and how to check someone's reputation, thus the two main subjects for building trust. Further will be investigated if there are some good initiatives on the market to tackle these problems, such as ws-security, ws-privacy, ws-trust, liberty, openid, jyte, etc.

Endpoint security

The Endpoint security process is responsible for providing the means to establish inherent trust levels between endpoints, with the intention to create a situation where all the devices involved in a transaction meet the criteria of trust for that transaction. At the moment, many Endpoint security or Network Access Control solutions exist. However, most of these solutions were not designed to interoperate with other solutions and they lack the ability to verify all network devices. Most solutions provide only Endpoint Security for PCs running certain Operating Systems.

Several Jericho Forum Commandments[1] refer to the Endpoint Security process.

- The second Jericho Forum Commandment states that "Security mechanisms must be pervasive, simple, scalable & easy to manage";
- The fifth commandment states that "All devices must be capable of maintaining their security policy on an un-trusted network";
- The seventh commandment states that "Mutual trust assurance levels must be determinable".

These commandments require a solution where every device connected to a network should be able to participate in the Endpoint security process. This means that a universal standard should exist that governs agent behaviour and interactions. For enabling secure devices to function in a possibly insecure network, these must be able to maintain their security policies. Consequently, this implies the existence of a solution that can monitor devices' status, can act upon it, essentially requiring agents installed on devices. Endpoint security research topic is interconnected with several other research topics within Jericho Project. The Authorization process within Jericho network will be dependent on the Endpoint security process for providing authorization information. In addition, in Jericho networks, the Accounting process will be used to handle information gathered by the Endpoint Security process.

The scope of the research is limited by the trust broker and other network services. The goal of this research is the establishment of a functional model for Jericho Project. In order to do this, the following steps have to be researched:

- Determine the role of Endpoint Security within the Jericho network
- Determine logical requirements for Endpoint security process
- Determine interaction requirements with research topics within Jericho Project
- Determine technical requirements of Endpoint security process
- Compare the established requirements with currently available solutions
- Recommend currently available solutions

Possible results of this research include the creation of a new, universal standard which can be applied to Endpoint Security, the establishment of a communication channel with Endpoint solution companies in order to Jericho-enable future versions of their products and the description or actual implementation of an available Endpoint security solution in a prototype environment.

[1] www.opengroup.org/jericho/commandments_v1.1.pdf accessed May 2007

Data classification and Information Leakage[1]

According to the 9th Jericho Forum Commandment, access to data must be controlled, in order to establish who may and who may not access the data. Firstly, the data should be encrypted for providing protection against entities that do not have yet the corresponding access level. Looking at Jericho Forum Commandments you can clearly see that there is a need to find an ideal way to classify data in order to prevent information leakage. The commandments that make clear there is a need to classify are:

9. *Access to data should be controlled by security attributes of the data itself*
10. *Data privacy (and security of any asset of sufficiently high value) requires a segregation of duties/privileges*
11. *By default, data must be appropriately secured when stored, in transit and in use*

Access to data: according to the 9th Jericho forum commandment, access to data must be controlled, in order to establish which entities may and which may not access the data. Firstly, the data should be encrypted in order to keep entities, that do not have yet at least the corresponding access level or that are not authorized, outside. Then the entities which may access the data have to be authorized. This can be done by the author but the problem is that it is way to much time consuming. It is a lot easier for the author to attach a group-status to data e.g. public, non-confidential or confidential. Per group-status are agreements like who may access the data and what the level of encryption must be. When the author uses group-status it is still time consuming. Data privacy (the 10th commandment of the Jericho Forum): To maintain the privacy of data, data will have an authentication level. Then the data can only be accessed by an entity when it is authorized and has at least the authentication level that is needed. Entities that are logged in get an authentication level, when an entity is logged in using only Name + Password its authentication level will be low, the more keys and/or certificates are used the higher its level is. A person that uses biometrics to log in will get the highest authentication level; this is because this the strongest type of authentication and can hardly be forged nowadays.
Appropriate secured data: According to the last commandment of the Jericho forum data should always be appropriately secured when stored, transmitted and used. In order to secure data appropriately it's again necessary to classify it first. After that the right level of security can be applied. The aim of this research is to find a way to automatically classify data in such a way that it can secure itself. This means that the data much is incapable of automatically authorize entities and attach the right level of authentication and security. Data classification & information leakage integrate with other parts of the Jericho Project research topics. It is interconnected with End-to-end encryption, Authentication and the Trust Broker research topics.

- When data is automatically attached with a level of security End-to-end encryption will decide how it must be encrypted.
- In cooperation with the Authentication process, the levels of authentication are determined.
- When an entity wants to access data, the data will request to the Trust Broker to verify that the entity is who it claims to be.

[1] Remco van Marle, Data classification & Information Leakage in Jericho Project, Capgemini, 2007

End-to-end encryption

In the context provided by Jericho Project, in this book we focus on exploring and proposing valid solutions for protecting data in transit, namely on investigating end-to-end encryption. The aim is to ensure the integrity and confidentiality of the data. In order to do this, the entities need to communicate securely and the transferred data has to be encrypted. Apart from privacy and integrity, for achieving secure communications other requirements have to be fulfilled as well: establishing a secure channel, the entities need to be authenticated, the source of the messages have to be authenticated as well, non-repudiation, accountability. Within the scope of Jericho Project, the goal of end-to-end encryption research is to investigate the possibilities offered by cryptography for designing and implementing suitable security protocols within Jericho networks for achieving secure communications. The starting point for this research is based on Jericho Forum Commandment[1] number 4 that states the following: "Devices and applications must communicate using open, secure protocols". As stated in Jericho Forum Position Paper "Enterprise Information Protection & Control" (Digital Rights Management)[2], in a de-perimeterised world it is generally easier to provide granular levels of data protection, the closer the protection mechanism is to the data. Moreover, the security of the data must reside with that data if it is to be adequately protected. Jericho Forum Commandment number nine states that "Access to data should be controlled by security attributes of the data itself" while commandment number eleven states "By default, data must be appropriately secured when stored, in transit and in use". The focus of this book is to offer protection of data in transit. For achieving these objectives, cryptographic mechanisms will be investigated and thoroughly analyzed in this book with the scope of selecting the adequate for protecting data in transit. However, members of Jericho Forum acknowledged that there is a need for open and interoperable standards for specifying these principles in order to achieve adequate data protection in all its states. Open standards ensure that the security principles can be thoroughly reviewed. Moreover, an open, inherently secure protocol is needed for communications that involve transferred data between the enterprises' servers and other entities in the system.

Scope of end-to-end encryption within Jericho Project

The research on end-to-end encryption for secure communications aims to offer recommendations for Jericho networks regarding the following aspects:

- establishing the requirements for secure communications within Jericho networks
- investigating a range of security protocols that can be used for end-to-end encryption for Jericho Project
- choosing the most adequate cryptographic algorithms and primitives, in terms of security offered and performance, that should be used for security protocols that offer end-to-end encryption in Jericho networks
- defining a roadmap for the implementation of the proposed solutions for Jericho networks.

Steps to be followed for secure communications

[1] www.opengroup.org/jericho/commandments_v1.1.pdf accessed May 2007

[2] www.opengroup.org/jericho/EIPC_v1.0.pdf accessed May 2007

In order to provide end-to-end encryption for the data in transit, in the context of Jericho Project, a number of steps have to be followed. Firstly, the entities involved in the communication have to be authenticated in a handshake protocol. Secondly, there will be established a secure connection between the entities, and a secure session will be set up for transmitting the content securely. This step is being performed or not, according to the output of authentication, authorization processes. Then, the adequate cryptographic primitives are chosen for achieving, further in the communication process, certain security services (e.g. confidentiality, integrity, authentication of the source of messages, non-repudiation). Also, the most suitable ways of distributing the keys are chosen and an agreement is reached. Typically, the public-key algorithms are used for secure key exchange, while the symmetric-key algorithms are used for encrypting the data in transit. Moreover, for the research on end-to-end encryption in the context of Jericho Project, the sources of the transmitted messages are authenticated, so the aim is to achieve authenticated encryption for data in transit. In order to achieve authenticated encryption for the data in transit we will investigate the security services provided by different security protocols, such as SSL/TLS, IPsec, XML encryption, and we will recommend the most suitable cryptographic primitives that can be used in the design of these protocols in the research context of Jericho Project. In Jericho Project, the end-to-end encryption research topic is inter-connected with the following modules that are part of Jericho Project research also: data classification, trust broker, accounting and authentication modules. The data is previously classified before encryption occurs. So, certain security protocols and/or cryptographic primitives may be given preference for implementing end-to-end encryption depending on the type of data (e.g. highly sensitive, confidential, public etc.). In addition, entities have to authenticate before transferring data. Moreover, logs of the authentication process, of the encryption process will be kept for further checking. The trust broker deals with the establishment and distribution of the encryption keys, the secure storage of the encryption keys.
The end-to-end encryption research aims to provide the following results within Jericho Project:

- The logical requirements for secure communications in Jericho network
- Investigation and comparison of different secure protocols of interest for secure communications
- Overview and comparison of different cryptographic primitives that can be used for designing security protocols for secure communications
- Explore which possible solutions may be used for implementing secure communications, respectively end-to-end encryption, for Jericho network
- A practical recommendation concerning the possible solution(s)

Discussion – overview of roadmap for Jericho Project

For providing a new security architecture, it is of interest for Jericho Project to take the following things into consideration as well:

- Existing threats on networks
- Choosing appropriate security services to protect network applications, data in transit, stored or in use, devices, and also correctly incorporating these services into the network
- Selecting cryptographic algorithms to implement certain security services (e.g. confidentiality, authentication of the sources of messages)

- Employing existing security protocols—such as SSL/TLS or IPsec—when appropriate, or developing new protocols when necessary
- Implementing security within the appropriate network layer
- Balancing security and performance when choosing between different technologies, security protocols
- Finding ways to classify data automatically
- Addressing issues regarding identity management, trust brokers, digital rights management, network quality of service, business partner risk management

3. Secure Communications in the context of Jericho Project

More and more businesses and individuals use Internet for sharing sensitive information and for conducting online transactions that should stay private among them. In the literature (Ramachandran, 2002; Stamp, 2006) it is asserted that the purpose of security is to enable valid communication, preferably in a transparent manner for the users. So, in the context of Jericho Project, all invalid communication – whether unauthorized, unauthenticated, forged or unwanted – should be blocked and prevented from occurring in Jericho networks. In order to achieve these security goals, certain steps should be implied: all the principles should be authenticated before initiating the communication in Jericho networks, further, they should be adequately authorized, and all the communication should be encrypted, verified for integrity and monitored. For accomplishing these objectives within the scope of Jericho Project, sound security policy should be defined and enforced in Jericho networks. Further, robust and thorough security protocols should be used for designing security solutions for secure communications in Jericho networks.

In this chapter, we present the rationale for secure communications and end-to-end encryption in Jericho Project. Moreover, the security services and security mechanisms for achieving secure communications within Jericho networks are explored. Besides this, the security protocols intended to implement the security mechanisms within the scope of this book are analysed in detail. Finally, there will be made suggestions regarding new ways of implementing security services and achieving protection against attacks. Next, the focus will be on exploring security protocols that are intended to offer these security services for secure communications – end-to-end encryption – in the context of Jericho Project. For achieving these security services, we will use the means offered by cryptography. Typically, cryptographic algorithms or primitives provide certain security services. Mao (2003) acknowledged that a challenging task in applied research on cryptography and cryptographic protocols is to build high quality security services from practical and available cryptographic algorithms and primitives. In our exploration of cryptographic (secure) protocols for secure communication in the context of Jericho Project, we will specify in detail the services that are intended to be deployed by the protocols and the mechanisms to achieve these security services. The aim of this investigation is to choose the most adequate cryptographic primitives and algorithms to be used within the security protocols for accomplishing the goal of secure communication in Jericho Project. Hartman et al. (2003) specified that two approaches can be employed for protecting the transferred data: connection-oriented or message-oriented. Connection-oriented mechanisms protect the messages while they are being transmitted between systems. For the data in storage, application or operating system mechanisms are used for protection. Message-oriented approaches protect messages in transit or in storage. Connection-oriented solutions include SSL and IPsec. Message-oriented solutions include XML Encryption and S/MIME. In context of Jericho Project, we will investigate the real-time protocols SSL/TLS and IPsec, in which the communicating parties negotiate and interact for mutual authentication and for establishing session keys for cryptographic protection of the communications. Both protocols, SSL and IPsec are used to securely transmit data from endpoint to endpoint.

A connection is established between two communicating parties. The transferred data is encrypted, in this way being protected in transit. So, while the data is being transported, it is protected from eavesdropping and modification. But, at the endpoint, the data is exposed. Consequently, in some cases connection-oriented protection may need to be augmented with other security services for offering the desired level of protection in Jericho networks. Finally, we investigate also XML Encryption for protecting data in transit in the context of Jericho Project. An advantage of XML Encryption is that it allows to encrypt also certain parts of a message, and not necessarily the whole message. For instance, only the sensitive information in a message can be encrypted, while the public information can remain in clear. However, the entire message can be signed by the sender for integrity checking. In this research we will explore also the possibilities offered by XML encryption for achieving secure communications in Jericho networks.

Requirements for secure communications for Jericho Project

According to members of Jericho Forum, businesses require wider collaboration between companies outside the enterprises' perimeters due to the explosion of pervasive, fast, reliable, and cheap Internet connectivity. In the White Paper – Business rationale for de-perimeterization[1], Jericho Forum members mentioned the reasons for a new security architecture based on de-perimeterization. Nowadays there is an increasing trend of business collaboration that involves the alignment of business activities and processes with other businesses to create mutual benefit.

There are certain aspects that indicate that companies are going toward a de-perimeterized security environment[2]:

- Businesses become more integrated and the collaboration relationships extend over the organizational perimeter
- Business demands to interconnect systems directly where B2B relationships exist
- There is a need to have a very good network connectivity and access to all organisations with whom an organization has business relationships
- Distributed / shared applications across business relationships
- Increasing use of Web services
- Increasing inability of traditional firewall and other network perimeter controls to protect the sensitive data against threats

The drivers for the new security approach are derived from the business needs, as well from the technical needs. Further, we enumerate shortly the business and technical drivers that underpin the research of Jericho Project.

[1] http://www.opengroup.org/jericho/Business_Case_for_DP_v1.0.pdf accessed May 2007
[2] http://www.opengroup.org/jericho/Business_Case_for_DP_v1.0.pdf accessed May 2007

Business Drivers

- collaborative business environment for electronic commerce
- demand for low cost collaboration and commerce over open networks and interfaces
- the connectivity requirements increase and demand for flexibility and adaptability
- new business models based on electronic transactions and mobile users
- relationships and partnerships within and among organizations
- increasing number of online collaboration and transactions among business entities; new work patterns

Technical Drivers

- protection needed closer to the application and the data
- need for protecting the information itself (not to the storing/transmitting medium)
- need for security of devices and data; the protection of the network is not the main focus anymore
- more sophisticated and faster online threats and new kinds of attacks

In Figure 2 there are illustrated the stages of computing history that lead towards a full Internet-based collaboration model among organizations, and finally to a full de-perimeterized network architecture.

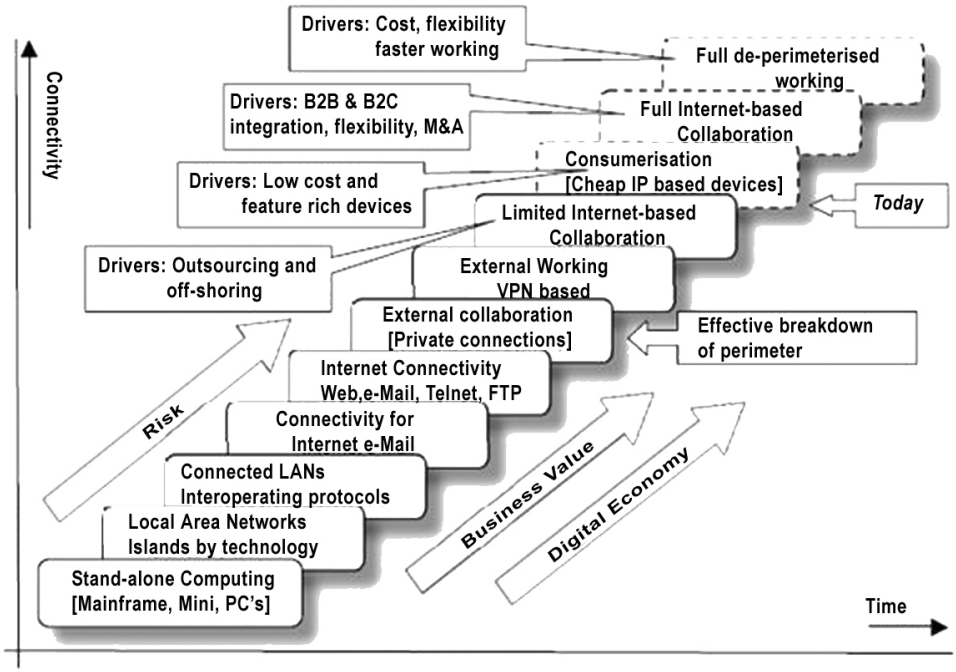


Figure 2: Computing history and business rationale for de-perimeterization (adopted from Jericho Forum White Paper Business rationale for de-perimeterization[1])

Undoubtedly, companies need access to the "network", namely to the Internet, and need to share information, to establish secure communications over Internet. But

[1] www.opengroup.org/jericho/Business_Case_for_DP_v1.0.pdf accessed May 2007

transferring data over Internet is vulnerable to a series of attacks. The number of possible users, the type of authentication (sometimes anonymous), locations, and the opportunity for error introduced by the global complexity of the Internet, all contribute to this vulnerability. Security of communications protects information that is transmitted over insecure networks, specifically the Internet. Following, we aim to provide a systematic overview of security requirements of secure communications in the context of Jericho Project. Prior to identifying the properties of secure communications across Internet, we enumerate some potential threats. Next, we explore the security mechanisms intended to defend against attacks that attempt to violate desired properties of secure communications. Consequently, we discuss means to implement these security mechanisms in open, secure protocols that accomplish the requirements for secure communications in Jericho network. We will investigate the existent solutions that could be implemented in Jericho network for achieving end-to-end encryption. The most widely used means to secure data against tampering and eavesdropping, the Secure Sockets Layer (SSL) and its successor, the Transport Layer Security (TLS) protocol are discussed, as well as Internet Protocol Security (IPsec).

The importance of security has increased after the incidence and gravity of attacks have boosted on Internet. Stallings (2005) presented the following attacks that can be identified in the context of communications across a network[1] :

- *Disclosure:* Release of message contents to any person or process not possessing the appropriate cryptographic key.

- *Traffic analysis:* Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

- *Masquerade:* Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an attacker that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or non-receipt by someone other than the message recipient.

- *Content modification:* Changes to the contents of a message, including insertion, deletion, transposition, and alteration.

- *Sequence modification***:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

- *Timing modification:* Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual

[1] We consider in this book that the network is Internet

message (e.g., datagram) could be delayed or replayed.

- *Source repudiation:* Denial of transmission of message by source.
- *Destination repudiation:* Denial of receipt of message by destination.

The first two types of attacks can be dealt with by applying measures that achieve message confidentiality. Against the attacks that modify the transferred messages, mechanisms for message authentication are implied. In Chapter 4 we describe in detail the mechanisms for achieving confidentiality and message authentication. These kinds of attacks, along with increasing cyber-crime, have encouraged the Jericho Forum's members to find new means and methods for protecting the networks. Based on the Jericho Forum's principles, in Jericho network every component will be independently secure, requiring systems and data protection on multiple levels, using a mixture of: encryption, inherently-secure computer protocols, inherently-secure computer systems, data-level authentication. Furthermore, we aim to establish a framework for the consequent exploration of both the security protocols and cryptographic mechanisms that can be used for achieving the security services for secure communications in Jericho Project. Based on Jericho Forum Commandments and the Position Papers proposed by Jericho Forum[1], a suite of security services should be accomplished for deploying and securing a Jericho network solution. Stallings (2005) explains based on standard definitions what a security service is. A security services can be defined as a service provided by a protocol layer of communicating open systems that ensures adequate security of the systems or of data transfers[2]. Another definition is provided in RFC 2828[3], in which a security service is defined as a processing or communication service that is provided by a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms. Within the scope of this book, we propose for Jericho networks, the attainment and enforcement of the following security features (services) in order to design and implement secure communications:

[1] http://www.opengroup.org/jericho/publications.htm
[2] http://fag.grm.hia.no/IKT7000/litteratur/paper/x800.pdf accessed May 2007
[3] www.ietf.org/rfc/rfc2828.txt

Confidentiality

Confidentiality refers to the protection of transferred data against attacks conducted by unauthorized entities. Transferred data should remain private and should be read only by the intended recipients. So, the communications should be kept private from all parties except the ones entitled to receive them. Basically, confidentiality prevents unauthorized disclosure of sensitive information. Raina (2003) specified that good security practice demands that all communication is encrypted and kept private, so no information can be revealed to intruders who conduct attacks on the network. The standard mechanism for enabling confidentiality is encryption. In the context of Jericho Project, we will focus in this book on the protection of transferred data over Internet. According to Lai (2002), confidentiality services based on encryption provide limited protection against traffic flow analysis. Traffic flow analysis represents a process by which an attacker tries to deduce valuable information by monitoring the frequency and amount of network traffic flowing between two communicating parties. While some security protocols mask the contents of a message, and conceal also its source and destination, other protocols just mask the content of the transmitted message without concealing the header information. But there are other techniques, such as traffic padding and routing control that can be used to conceal message length and frequency of network traffic. Traffic padding represents the injection of spurious traffic into a network to hide actual usage patterns. Routing control involves directing traffic along a particular path between sender and recipient in order to reduce exposure to eavesdropping.

Authentication

Authentication can be subdivided into entity authentication and data origin authentication. Entity authentication refers to the ability to verify the identities of all entities involved in a message transmission. It ensures that the participating entities in a communication process are the ones who they claim to be. Typically, it is provided for use at the establishment of (sometimes at different times during the data transfer) a connection. This security services is intended to offer protection against attackers who can impersonate authenticated entities and perform either a masquerade or an unauthorized replay of a previous connection.

Data origin authentication refers to the corroboration that the source of data received is the one claimed. This service must assure that the connection is not interfered so that a third party can masquerade as one of the two legitimate parties for the purposes of unauthorized transmission or reception. A system implementing the authentication property assures the recipient that the data is from the source that it claims to be. Assuring this service involves binding a set of credentials to a message; these are verified by the receiver upon the receipt of the message (Lail, 2002). Mechanisms that provide data origin authentication are keyed message digests and digital signatures, which offer also integrity protection.

Non-repudiation

Non-repudiation service refers to the prevention of denial by an entity (the sender or recipient of a message) that has taken a particular action, such as sending or receiving a message. With this service, the receiver can prove that the alleged sender in fact sent the message and vice versa. For instance, in the case of transactions between a service provider that offers an online service and the customers, non-repudiation hinders both the customer and the service provider from credibly denying that a transaction occurred at a particular date and time. When a message has been transferred, the sender can prove that it has been received. Similarly, the receiver can prove that the message has actually been sent. The primary mechanisms for achieving non-repudiation are digital signatures in combination with timestamps (Lail, 2002; Stallings, 2005). Timestamps bind a transaction with the time and date when it occurred. In fact, non-repudiation has more legal implications than technical ones. Lail (2002) pointed out that non-repudiation encompasses an entire set of policies and procedures for establishing and enforcing trusted communication between different entities.

Integrity

Integrity assures that transferred messages are received as they are sent, with no duplication, insertion, modification, reordering, or replays. Also deletion or destruction of data is included in this service, so all the transferred data should arrive to the receiver (Stallings, 2005). So, this service prevents the unauthorized alteration or destruction of transmitted data by unauthorized entities. Typically, this security service is obtained through the use of hash functions (message digests).

Access control

In the context of network security, access control is the ability to limit and control the access to host systems and applications via communications links. In order to achieve this goal, each entity trying to gain access must first be identified, or authenticated, so that access rights can be issued adequately to the individual.

Availability

Availability characterizes a system whose resources are always ready to be used. In the context of communications over Internet, this means that whenever information needs to be transmitted, the communication channel is available and the receiver can cope with the incoming data. This property makes sure that attacks cannot prevent resources from being used for their intended purpose.

All these security services or properties that we described are required in order to ensure secure communications in Jericho network. Following, we mention a series of important objectives that can be accomplished by the means of security services (Lail, 2002), and that apply perfectly for secure communications in Jericho network as well:

- Ensure that confidential information is not viewed by unauthorized entities
- Guarantee that a message cannot be altered without detection on its way from sender to recipient; the data is not modified in transit
- Verify that the data was not modified

- Authenticating the identity of the entities that are communicating and of the source of a message
- Preventing the sender of a message from being able to deny his/her actions
- Determining whether a user has rights to certain resources, and ensuring that only authenticated users gain access
- Making data and network applications and services available to users who request access to them

In order to achieve these security services for secure communications in Jericho network, there are used certain tools named security mechanisms. Typically, there are multiple mechanisms for implementing each security service. Thus, one of the challenges in designing a secure system is selecting the most appropriate mechanisms. This is also our purpose in this research, to choose the most appropriate mechanisms for implementing secure communications services for end-to-end encryption in the context of Jericho Project.

In the following sub-chapters chapters, we will investigate a series of security mechanisms provided by cryptography that can be used for implementing security services for end-to-end encryption in Jericho network. Such mechanisms include: encipherment mechanism, digital signature mechanisms, data integrity mechanisms, authentication mechanism etc. In this book we will deal further with the selection of the appropriate cryptographic mechanisms for end-to-end encryption in Jericho network. The selection of certain cryptographic mechanisms for achieving a security services is based on several factors. Such factors include: compatibility with legacy applications, complexity of computation, and intellectual property issues. Typically, different security mechanisms can be used to enforce the security properties defined in a given security policy. Depending on the anticipated attacks, different means have to be applied to satisfy the desired properties. The different measures against attacks can be divided into three classes of security mechanisms: mechanisms for attack prevention, for attack avoidance and for attack detection (Kruegel, 2005).

Attack prevention mechanisms are a class of security mechanisms that contain ways of preventing or defending against certain attacks before they can actually reach and affect the target. With respect to secure communications, an essential mechanism in this category is access control, an instrument that can be applied at different levels such as the operating system, the network or the application layer. Basically, access control mechanism limits and regulates the access to critical resources. This is done by identifying or authenticating the party that requests a resource and checking its permissions against the rights specified for the demanded object. Within Jericho Project, this mechanism is investigated in separate research topics that deal with identity management - AAA framework. So, this signifies that secure communications, more specifically end-to-end encryption, in Jericho networks interact and depend on the specification and implementation of access control mechanism. Consequently, before implementing secure communications as desired in Jericho Project, firstly the entities have to be adequately authenticated and authorized before having access to the data they want to access. Another relevant element in the set of attack prevention mechanisms is *system hardening*. System hardening is used to describe all steps that

are taken to make a computer system more secure. In the context of Jericho Project this mechanism is dealt with in the research topic focused on *endpoint security*.

Attack avoidance mechanisms assume that an intruder may access the desired resource but the information is modified in a way that makes it unusable and invaluable as well, for the attacker. Based on the principles of these mechanisms, the information is pre-processed at the sender before it is transmitted over the communication channel and post-processed at the receiver. While the information is transported over the communication channel, it resists attacks by being almost useless for an intruder. Nevertheless, attacker can still conduct attacks against the availability of the information, as they could still interrupt the message transmission. Moreover, during the processing step at the receiver, modifications or errors that might have previously occurred can be detected by means of integrity checks.

The most important mechanism in this category is cryptography which is defined by Schneier (1996) as the science of keeping messages secure. Within the scope of this book in Jericho Project, we will deal with the security mechanisms and, besides this, we will discuss the implementation of security protocols based on these security mechanisms. Attack avoidance and intrusion detection mechanisms aim to offer protection against the situations in which an attacker might have obtained access to the desired target and succeeded to violate the corresponding security policy. Within the context of secure communications, mechanisms in this category are based on the unrealistic assumption that most of the time the information is transferred without interference. Attack detection mechanisms have the task to report possible intrusions or attacks that were conducted with the scope of hindering secure communications. It would be also desirable to identify the exact type of attack. The most important mechanism of the attack detection category is represented by intrusion detection systems. Anyway, it is out of the scope of this book to investigate in more detail the protection means offered by these mechanisms for achieving secure communications.

The relationship between security services and mechanisms (adapted after Stallings, 2005):

Mechanisms					
Service	Encipherment	Digital Signature	Access Control	Data Integrity	Authentication Exchange
Entity Authentication	Y	Y			Y
Data origin Authentication	Y	Y			
Access control			Y		
Confidentiality	Y				
Traffic flow Confidentiality	Y				
Data integrity	Y	Y		Y	
Non-repudiation		Y		Y	
Availability				Y	Y

In the documentation “Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0” there is described a decision matrix for choosing the adequate security mechanisms for assuring message protection for transmitted data over the Internet. In Table below we illustrate a modified version of this message protection decision matrix. The decision matrix lists the security considerations related to message protection and specifies how they are supported by the proposed security mechanisms.

Security Consideration	Data Integrity	Data Origin Authentication	Data Confidentiality
Verification if the contents of a message were not altered in transit.	Allows verification that a message has not changed in transit.	Supports the ability to verify that a message has not changed in transit and verify the origin of a message.	Encryption does not prevent the contents of a message from being altered.
Verification of the data source authentication and integrity of the message (has not been altered in traffic)	Allows verification that a message has not been changed, but this does not necessarily imply that the receiver can verify the source of the data.	Supports the ability to verify that a message has not changed in transit and verify the origin of a message.	Encryption does not prevent the contents of a message from being altered.
Restriction of the access to the contents of a message to authorized users only.	Does not provide the ability to protect message contents from unauthorized users.	Does not provide the ability to protect message contents from unauthorized users.	Confidentiality can be used to encrypt the contents of a message so that only authorized users can view the message contents.
Authentication is implemented based on shared secret between the entities participating in the communication. Prevention is required against attackers who want to recover the shared secret	Generating signatures based on shared secrets that may have low entropy leaves the message vulnerable to offline cryptographic guessing attacks; instead, direct authentications mechanism can be used	Generating signatures based on shared secrets that may have low entropy leaves the message vulnerable to offline cryptographic guessing attacks; instead, direct authentications mechanism can be used	Encryption combined with data integrity and data origin authentication can be used to protect the shared secret.
Implementation of message replay protection for preventing an attacker from maliciously replaying messages. Replay detection depends on the ability to uniquely identify messages.	This option is often implemented using a hashing function that provides a unique identifier that can be used to determine if the same message is received multiple times.	This option is often implemented using a hashing function or digital signature that provides a unique identifier that can be used to determine if the same message is received multiple times.	Not applicable.

So, we summarize that in order to protect the data in transit in Jericho networks, at least three security services should be provided: integrity, confidentiality and data origin authentication. The content of the transmitted messages remains private, is not altered and can be verified against tampering upon receiving. For acquiring these security services, typically, cryptographic mechanisms (used in security protocols) are employed. In order to establish secure communications, apart from using cryptography, several steps must be performed in a specific order. For building secure communications is essential to establish a secure session. By secure session we imply that it uses cryptography and other monitoring and mitigation processes in order not to allow the leakage of any information and to protect both the server and client from any exposure. Moreover, identification, authentication, and all other access-level decisions

on the information that exist at the application level must be performed. Besides all these, all of the protocol level information should be logged for auditing purposes.

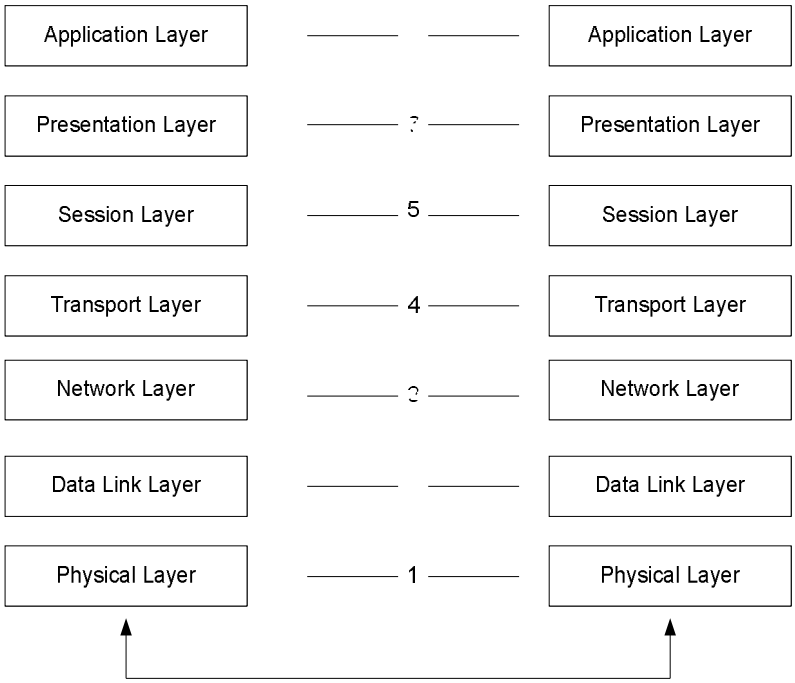
For creating a secure communication between two entities, each entity is required to perform the following:

- Send a connection request. One party initiates the contact, and the other must respond
- Negotiate communication and cryptographic terms of engagement
- Authenticate the peer entity
- Manage and exchange of the session keys
- Renegotiate keys on request
- Establish data transfer properties such as encryption or protection
- Manage errors by enabling the use of exceptions, communicating alerts, or sending error messages
- Create audit logs
- Close connections on successful completion or on fatal errors
- Re-establish closed connections based on bilateral agreement from the entities

Overview of Open Systems Interconnection (OSI) Model

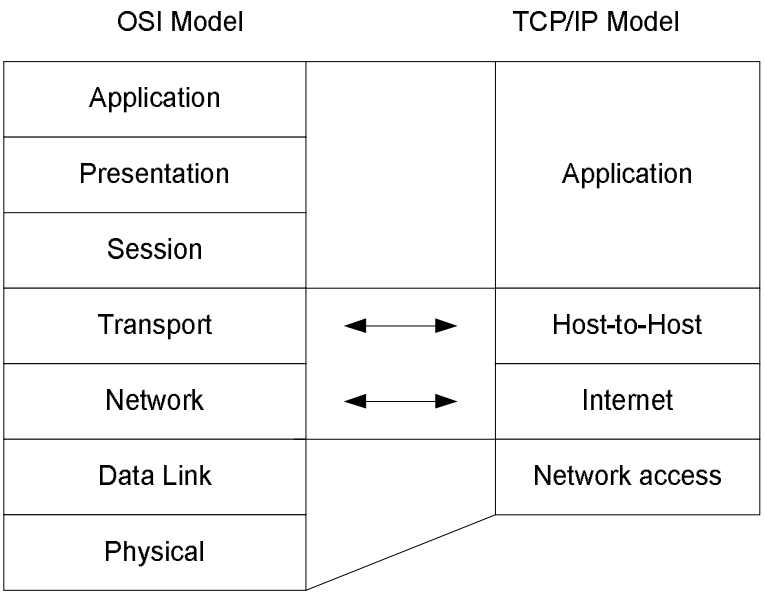
The International Standards Organization (ISO) introduced the seven-layer Open Systems Interconnection (OSI) network protocol stack as a model for network communications. The OSI reference model segments the networking tasks, protocols, and services into different layers. Each layer in the stack has its own responsibilities and functionalities regarding how two computers communicate over a network. OSI reference model is a layered model for understanding and implementing computer communications and computer network protocols. Further on, we will present which the vulnerabilities of each layer are and what protocols can be used for providing secure communications. Each layer in OSI model has a special interface (connection point) that allows it to interact with three other layers. A layer can receive communications from the interface of the layer above it, can transmit communications to the interface of the layer below it, and can establish communications with the same layer in the interface of the target packet address. The control functions, added by the protocols at each layer, are in the form of headers and trailers of the packet.

Figure 3: The OSI Reference Model:



The Transmission Control Protocol/Internet Protocol (TCP/IP) is a suite of protocols that governs the way that data travels from one device to another.

Figure 4: The differences between the OSI and TCP/IP networking models.



Each OSI layer has a set of functions to perform for enabling the data to travel from a source to a destination on the network.
The 7 layers of the OSI Reference Model (adapted after de Laet & Schauwers, 2004):

OSI LAYER	Name	Description	Vulnerabilities
Application	Network processes to	Telnet, FTP, rlogin, Windows,	Examples include: e-mail bombs and

	applications	Mac OS, UNIX, HTTP, SNMP, RMON[1], DNS, whois, finger	spam; Trojan horses; viruses; unauthorized access to key devices; brute force attacks; browser holes; malicious Java, active-X, or CGI exploits; reconnaissance and mapping; control daemons; holes; keyloggers;
Presentation	Data representation	ASCII, EBCDIC[2], HTML, Pict, wav	Unencrypted data formats can be viewed; compressed Trojan and virus files can bypass security; weak encrypted data;
Session	Interhost communication	NFS, SQL[3], RPC[4], Xwindows, Bind, SMB[5]	Traffic monitoring; root access;
Transport	End-to-end connections and reliability	TCP, UDP, SPX[6]	Exploitations using SYN flooding and TCP hijacking; spoofing; port scans; fragmentations
Network	Address and best path	IP, IPX[7], ICMP	Ping scans and packet sniffing; ARP poisoning and spoofing; DDos Smurf; IP spoofing; Tribe Flood Network; nuking
Data Link	Media access	MAC[8], LLC[9]	Reconnaissance and sniffing Frame manipulation; spoofing broadcast storms; ARP cache poisoning; misconfigured and failing NICs Stored attack robots (bots)
Physical	Binary transmission	Media, connectors, devices	Wire tap and sniffing; full network access and recon in a nonswitched LAN; natural disasters; power failure; theft etc.

1. RMON = Remote Monitoring
2. EBCDIC = Extended Binary Coded Decimal Interchange Code
3. SQL = Structured Query Language
4. RPC = Remote-procedure call
5. SMB = Server Message Block
6. SPX = Sequenced Packet Exchange
7. IPX = Internetwork Packet Exchange
8. MAC = Media Access Control
9. LLC = Logical Link Control

Each layer of the OSI model has certain vulnerabilities. Understanding the vulnerabilities of each layer and how different types of attacks can occur helps in assessing the risk and search for adequate security solutions. There are mechanisms for building reliable and secure communications at all layers of the OSI / TCP/IP model. Each of these mechanisms has its advantages and disadvantages. Protection provided at the application layer
(Layer 7) is application specific. Thus, the protection methods need to be reimplemented in every application on the host. By adding protection at the transport layer, application independence is gained. The implemented security mechanism might require running over a specific transport-level protocol. Secure Sockets Layer (SSL) runs over TCP because it is session-oriented and it requires reliable communication. Contact with the application is lost when protection is added at the network level. In

order to capture the user context, the network layer security mechanism must depend on a higher-layer interaction. This captures the user context called security association, and transfers it down to the network layer. Protection at the data link layer provides protocol-independent protection. Data link layer protection is expensive to deploy on a large scale because there is a need to protect every single link separately. In order to provide protection at the lower levels of the OSI model (Data Link and Physical), hardware protection units or dedicated private lines can be used for protecting a communication link. The Secure Sockets Layer protocol provides application and transport-layer security, and IPsec provides network-layer security. IPsec represents a framework of security protocols and algorithms used to secure data at the network layer. Essentially, the Transport layer is intended to provide reliable communications between two endpoints. Gregg (2006) acknowledged that both SSL and TLS protocols build upon the traditional functionality of TCP to provide confidentiality (by encryption) and integrity (via hashing and digital signatures).

If encryption is implemented on one layer, this means that the respective layer and all the layers above are protected. Protection implemented at the network layer it offers one of the most flexible solutions because it is media independent and at the same time, also, application independent. But nowadays, SSL becomes more popular because of its advantages over IPsec. If encryption takes place at the lower layers of the OSI model, for instance at physical and data layer, this is called *link-to-link encryption*[1]. If the encryption takes place at higher layers, it is called *end-to-end encryption*.

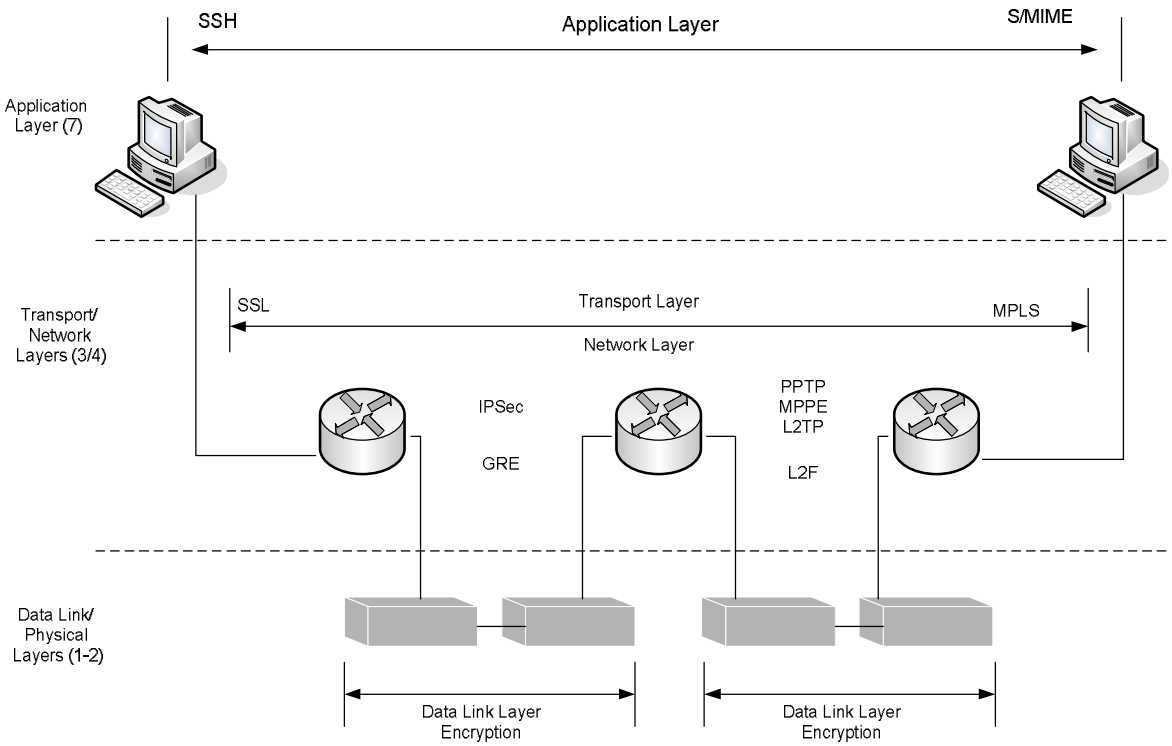


Figure 5: An example of encryption different layers, options of protection at different layers'' (adapted after de Laet & Schauwers, 2004):

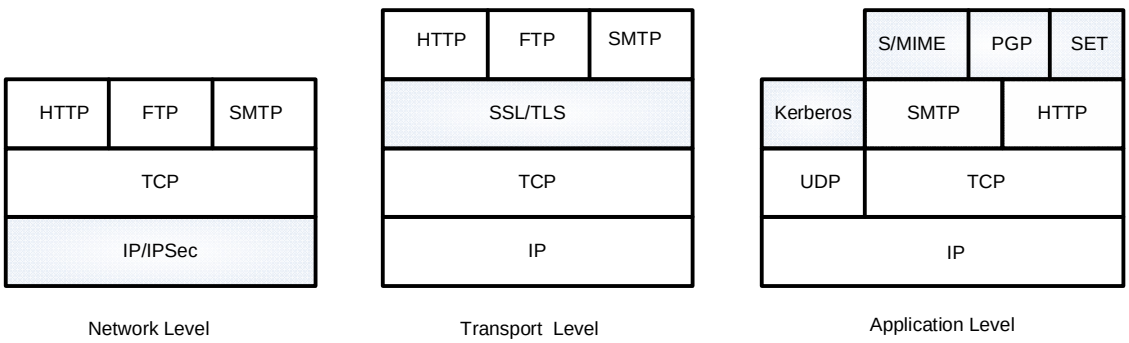
[1] In the literature the link-to-link encryption is also named link-by-link encryption or link encryption.

Security protocols

In the context of Jericho Project, for achieving secure communications there will be investigated security protocols and cryptographic primitives that provide different security services. In the current chapter we will explore security protocols that are of interest for Jericho networks and that can be used for different security purposes, such as entities authentication, privacy, integrity, authentication of the source of messages. We will focus our research on SSL/TLS and IPsec security protocols within the scope of this book. These protocols are also named cryptographic protocols because they perform security-related functions for which cryptographic primitives are used and applied. A cryptographic protocol can be defined as a sequence of steps that uses encryption and decryption to secure the communication between two or more computers on the network. Internet Protocol Security (IPsec is an extension of the Internet Protocol (IP) and provides a transport level secure communication solution. It provides security at the IP-enabled communications between two or more computers on a network. IPsec provides end-to-end security in network configurations such as client-to-server, client-to-client, and server-to-server. In the course of our future research for Jericho project we will investigate also other technologies and protocols for providing secure communications over the network (basically the network will be the Internet). In the position paper "The Need for Inherently Secure Protocols"[1] on Jericho Forum, they recommend also the use of: SMTP (Simple Mail Transfer Protocol), AS2 (Applicability Statement 2).

Further, in this chapter, we will present in detail two of the most used security protocols used in the real world for securing the communications. Firstly, we will present Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol that is used nowadays intensively for securing Internet transactions between web browsers and web servers. Secondly, Internet Protocol Security (IPsec) protocol will be described in detail. Moreover, we analyse the security flaws existent in these protocols and the possible solutions for fixing these security problems. Then we will make a comparison between these two protocols in order to see which could provide a better solution for Jericho Project, and in which cases.

In the figure below there are presented different security facilities in the TCP/IP protocol stack



[1] www.opengroup.org/jericho/Protocols_v1.0.pdf accessed March 2007

Figure 6: Relative Location of Security Facilities in the TCP/IP Protocol Stack (after Stallings, 2005)

IPsec protocol is transparent to end users and applications, and provides a general-purpose solution. Further, IPsec includes a filtering capability so that only selected traffic need incur the overhead of IPsec processing. Another option to provide a general-purpose security solution is to implement security above TCP (above the Transport Layer) by using, for instance, SSL or TLS protocol. Stallings (2005) specified that there are two implementation choices for this protocol: SSL/TLS can be part of the underlying protocol for providing full generality or SSL/TLS can be embedded in specific packages (e.g. Netscape and Microsoft Explorer browsers have options for this protocol, as well the Web servers). In the third picture in Figure 3.5 there are illustrated application-specific security services that are embedded within particular applications. These services can be tailored to the specific needs of a given application.

Next, we discuss the security protocols of interest for providing secure communications in the context of Jericho Project.

Secure Socket Layer and Transport Layer Security (SSL/TLS)

Nowadays, all businesses, most government agencies, and many individuals have Web sites and there are many facilities created on the Web for electronic commerce. But, Internet and the World Wide Web are vulnerable to all sorts of security attacks. Consequently, the demand for secure Web services is growing (Stallings, 2005). Plaintext messages are sent and received by Web services over standard Internet protocols (e.g. Hypertext Transfer Protocol (HTTP)). By default, HTTP does not employ data encryption for transfers between the Web server and the Web client (Komar et al., 2004). The transmitted messages can be intercepted by attackers, potentially viewed and even modified for malicious purposes, replayed or the communications between the entities can be even interrupted (Microsoft Corporation, 2005). Web services are offering the possibility for fast and flexible information sharing across Internet. Hartman et al. (2003) mentioned in their book that Web services enable access to data that previously could be found only on corporate networks and was accessible only by using specialized software. Besides this, the authors pointed out the risks of exposing sensitive and private data to security attacks such as interceptions, unauthorized reading and modification of the data, replay of messages, reflection attacks etc. Hartman et al. (2003) made some observations regarding the security of Web services that are in accordance with the de-perimeterization principles proposed by Jericho Forum. Thus, the authors acknowledged that in the world of electronic commerce, where all the players (e.g. customers, suppliers, remote employees, partners, competitors) are collaborating over the Internet, end-to-end security solutions should be designed and deployed for protecting the sensitive data. With reference to secure communications, end-to-end means that sensitive data is encrypted all the way on the communication channel between the user and the data.

Jericho Forum's members presented in one of their position papers[1] the principles for managing data privacy. In this paper it is specified that there are privacy problems related to all kinds of existing data (confidential corporation data, personal information etc.) and refers mainly to the Personally Identifiable Information (PII). It is recommended in the paper that the privacy information associated with data must be bound to (or reside) with that data. These goals can be achieved by developing and using open and interoperable standards, open and inherently secure protocols for protecting the communications, and by creating and deploying a trusted framework for collecting, exchanging and using data[2]. Actually, organizations must focus on building more secure applications from the ground up while protecting the data at all times an in all forms (in storage, in process and in transit). Further, we present in detail one of the most used security protocols deployed in the real world for securing the communications, and that can be used also in Jericho Project for providing certain security services that are of interest within the scope of this research. Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol is used nowadays intensively for securing Internet transactions between Web browsers and Web servers for securely transferring data over insecure channels (Internet), as well for other applications (e.g. mail or news applications, Telnet, FTP etc.). Consequently, we explore the security services and the security mechanisms included in this protocol, and analyse its applicability in the case of Jericho Project. The primary goal of the SSL/TLS protocol is

[1] http://www.opengroup.org/jericho/Privacy_v1.0.pdf accessed May 2007
[2] Idem 1

to provide privacy and data integrity for transferred data between entities. Because the security services provided by the protocol match some of the requirements for secure communication in Jericho network, we explore further the possibilities offered by this current protocol in the context of Jericho Research Project. In this sub-chapter we present SSL 3.0, TLS 1.1 and TLS 1.2 (proposed standard) protocols and analyse their implications in providing secure communications and end-to-end encryption in the context Jericho network.

Presentation

Secure Sockets Layer (SSL) protocol version 2 (version 1 was never deployed) was invented by Netscape and it provides reliable end-to-end communications over the Internet between two hosts. According to Kaufman et al. (2002), Microsoft improved SSL 2.0 by fixing certain security problems, and introduced a similar protocol known as Private Communications Technology (PCT). SSL 2.0 (second version of SSL protocol) provides authentication of the client and the server, while SSL 3.0 (third version of SSL protocol) provides all the features of SSL 2.0 along with increased security and efficiency. SSL 3.0[1] has extra functionalities for data compression; it uses ciphers for communications and also certificate chains. It is acknowledged in the literature (Thomas, 2000; Kaufman et al., 2002) that Netscape Communications developed the first three versions of SSL protocol with significant assistance from the web community and with public review. Transport Layer Security (TLS) protocol, developed by Internet Engineering Task Force (IETF) is based on the specifications of SSL 3.0 protocol with a few modifications (Rhee, 2003; Johnston & Piscitello, 2006). It provides confidentiality and integrity for sensitive data sent over the Internet. Hassler (2001) pointed out that even if TLS is somehow similar, but yet there are some difference (e.g. in the cipher suites that they use for achieving different security services) that make them not interoperable. TLS has three versions. Till March 2007 the most recent version of TLS was 1.1, and is described in RFC 4346[2]. TLS 1.1 incorporates some minor security fixes and clarifications. The current version of TLS is 1.2, and is described in an Internet working draft[3] that expires in September 2007. Typically, SSL/TLS is most commonly used to secure the channel between a browser and Web server, namely for Web communications and web-based transactions. Due to its successful application for securing Web communications, SSL/TLS is used with other applications as well apart from HTTP, such as mail or news applications, Telnet, FTP etc. (Thomas, 2000; Lail, 2002; Oppliger, 2002; Ramachandran, 2002; Komar et al., 2004; Gregg, 2006). In fact, any upper-layer protocol or application that relies on TCP can employ the security services provided by SSL/TLS. Lail (2002) specified that SSL/TLS protocol, in addition to securing web-based traffic, can secure the following as well:

- Vendor-proprietary communication protocols
- Connections between back-end servers within enterprise and B2B environments
- Connections between network devices, such as provisioning equipment and routers, and the remote management consoles used to administer these devices

[1] http://wp.netscape.com/eng/ssl3/draft302.txt accessed April 2007

[2] http://www.ietf.org/rfc/rfc4346.txt accessed April 2007

[3] http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc4346-bis-03.txt accessed April 2007

SSL/TLS forms an extra layer between the transport (TCP) and the application layer and this represents an advantage, because no modifications are needed in the software (Oppliger, 2002; Aoufi, 2006). With SSL/TLS protocol, secure connections between clients and server applications can be generated, mutual authentication or server authentication can be achieved in one established communication session. SSL/TLS also provides privacy and integrity of the data that client and server exchange. Typically, only the server is authenticated (its identity is ensured), while the client remains unauthenticated. Based on the requirements of Jericho Project, both entities should authenticate, so there is established a mutual authentication. Both the SSL and TLS protocols allow client/server applications to communicate in such a way that they prevent eavesdropping, tampering or message forgery. Shortly, SSL/TLS protocol consists of a set of messages and rules about when to send (and not to send) each one (Thomas, 2000). Snader (2005) summarized the steps of an SSL/TLS session. Basically, an SSL/TLS session has three stages: connection setup, data transfer, and connection teardown. Firstly, the encryption, authentication, and compression algorithms are negotiated; the identity of the server and, optionally, the identity of client is verified also, and a key exchange takes place. Secondly, the client and the server exchange application data. These exchanges are encrypted and authenticated to ensure that the data cannot be read by third parties (encryption) and that third parties cannot alter the data without detection (authentication). After ending the transmission of the data, one entity or both of them send a closure notification as an EOF. The closure notification is authenticated, so it cannot be forged by third parties. Also, Snader (2005) noted that the SSL 3.0 and TLS 1.0 specifications require that both sides send closure notifications, but in practice, this is often ignored, and only one side sends it.

SSL/TLS Properties

SSL has the following *security features* (Ramachandran, 2002; Cole et al., 2005):

- SSL/TLS works between the application and transport layers of the network protocol stack to ensure security of applications on the transport layer
- SSL/TLS provides private, reliable, and non-forgeable conversation between two communicating processes
- Basically, SSL/TLS provides client-side and server-side authentication, confidentiality (encryption of the messages) and message integrity
- While frequently associated with web-based transactions, SSL is not limited to securing the Hypertext Transfer Protocol (HTTP). Any upper-layer protocol or application that relies on TCP can employ the security services provided by SSL (e.g. news, email, FTP, Telent, NNTP[1], IMAP[2], IRC[3], and POP3[4]).

They can all be secured by layering them on top of SSL (the appended letter "S" in the corresponding protocol acronyms indicates the use of SSL). But Oppliger (2002) also mentioned that SSL has a strong client-server orientation and does not really meet the requirements of peer application protocols.

- The communications can travel over non-secure networks

[1] Network News Transfer Protocol
[2] Internet Message Access Protocol
[3] Internet Relay Chat
[4] Post Office Protocol

Due to the use of encryption in securing communications, the risk of man-in-the-middle attack can be considerably reduced because the attacker cannot decrypt the transmitted message in a reasonable timeframe. The authors pointed out that this benefit assumes that SSL/TLS is properly configured and used at both ends of communication.

Applications can use a unique port number for SSL/TLS-protected communications. So, when the SSL/TLS is used in Jericho networks for protecting the transfer of data, the firewall configuration should be changed in order to allow the encrypted traffic. Although this will allow SSL/TLS sessions to be established through the firewall, the firewall will not be able to analyze the contents of the SSL/TLS-encrypted packets. As a result, the firewall will be able to use only the origin and destination of the packet to determine whether to let packets through. The typical ports that various applications use for SSL/TLS are listed in the tabel below:

Protocol	**Standard Port**	**SSL/TLS Port**
Hypertext Transfer Protocol (HTTP)	80	443
Simple Mail Transfer Protocol (SMTP)	25	465
Post Office Protocol version 3 (POP3)	110	995
Internet Message Access Protocol (IMAP)	143	993
Network News Transfer Protocol (NNTP)	119	563
Lightweight Directory Access Protocol (LDAP)	389	636
Global catalogue queries	3268	3269

As we have already specified, roughly, SSL/TLS involves three stages:

- Peer negotiation for algorithm support
- Public key encryption-based key exchange and certificate-based authentication
- Symmetric cipher -based traffic encryption

SSL/TLS protocol makes use of cryptographic algorithms and primitives for implementing the security services that supports. Firstly, the two entities need to exchange "keying material" with each other (Snader, 2005). Typically, in this process of key exchange, the server is also authenticated (optionally the client as well). Secondly, the application data and other messages have to be encrypted by means of symmetric-key cryptography in the protocol. Several ciphers, both stream and block, are supported for this service. Finally, each transmitted record must be authenticated. A message authentication code is added to each record. In this chapter we mention the possible combinations of cryptographic primitives and algorithms supported in SSL 3.0/TLS 1.1/TLS 1.2, while in Chapter 4 will we examine them in detail and attempt to make recommendations regarding the most adequate to be used in designing effective and efficient security protocols.

SSL/TLS Session

SSL/TLS session is assumed to be relatively long, so that many connections can be derived from a session (Kaufman et al., 2002; Oppliger, 2002; Stallings, 2005). If an SSL session exists, then two entities share a symmetric key *K* that can be used further to establish new connections. Rhee (2003) defined an SSL session as being an association between a client and a server. The author described in detail the role of an SSL/TLS session. Sessions are created by the Handshake Protocol. In a session, a set of cryptographic security parameters are defined and they can be shared among multiple connections. Sessions are used in order to avoid the expensive negotiation of new security parameters for each connection. An SSL session coordinates the states of the client and server. The state is represented twice as the current *operating state* and *pending state*. When the client or server receives a *change cipher spec* message, it copies the pending read state into the current read state. When the client or server sends a change *cipher spec* message, it copies the pending write state into the current write state. When the handshake negotiation is completed, the client and server exchange *change cipher spec* messages, and they then communicate using the newly agreed-upon cipher spec. As described, Oppliger (2002), Rhee (2003) noted that an SSL session is stateful and that the SSL protocol must initialize and maintain the state information on either side of the session.

SSL/TLS Session State Information Elements (after Oppliger, 2002; Rhee, 2003; Stallings, 2005):

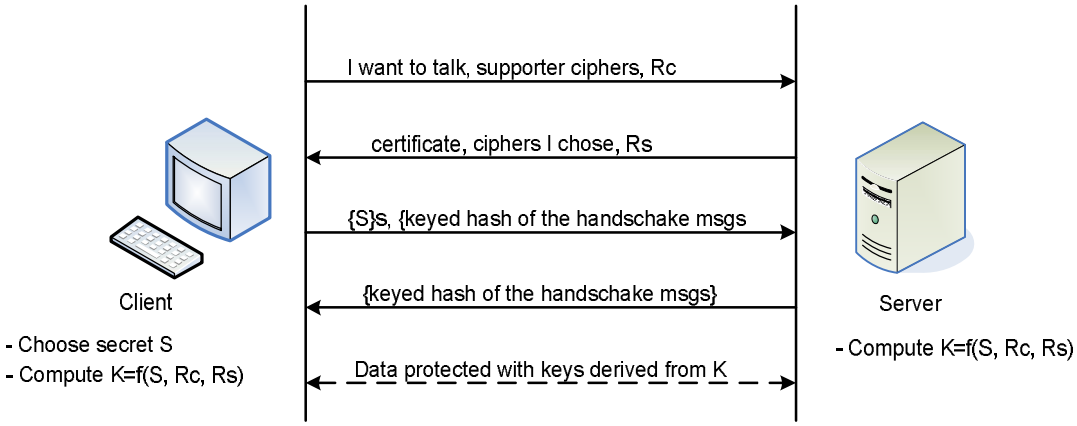
Session State Information Element	Description
Session ID	Identifier chosen by the server to identify an active or resumable session state
Peer certificate	X.509 version 3 certificate of the peer entity. This element of the state may be null.
Compression method	Algorithm used to compress data prior to encryption
Cipher Spec	Specification of the data encryption and MAC algorithms. It also defines cryptographic attributes such as the hash size.
Master Secret	48-byte secret shared between client and server. It represents secure secret data used for generating encryption keys, MAC secrets and IVs.
Is resumable	This is flag that indicates whether the session can be used to initiate new connections.

As we already mentioned, an SSL/TLS session can be used for several connections. A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. Rhee (2003) noted also that for SSL/TLS every connection is associated with one session. The corresponding connection state information elements are summarized in the table below (after Oppliger, 2002; Rhee, 2003; Stallings, 2005):

Connection State Information Element	Description
Server and client random	Byte sequences that are chosen by the server and client for each connection.
Server write MAC secret	Secret key used for MAC operations on data sent (written) by the server.
Client write MAC secret	Secret key used for MAC operations on data sent (written) by the client.
Server write key	Conventional cipher key used for data encryption by the server and decryption by the client.
Client write key	Conventional cipher key used for data encryption by the client and decryption by the server.
Initialization vector	Initialization state for a block cipher in CBC mode. This field is first initialized by the SSL Handshake Protocol. Thereafter, the final ciphertext block from each record is preserved for use with the following record.
Sequence number	Each party maintains separate sequence numbers for transmitted and received messages for each connection.

Oppliger (2002) noted that communicating parties may use multiple simultaneous SSL sessions and sessions with multiple simultaneous connections.

Figure 7: Simplified SSL/TLS



We will explain shortly the steps in establishing a secure session between a client and a server in SSL/TLS protocol (Kaufman et al., 2002; Stamp 2006):

- The client C initiates contact with the server S, sends also a list containing the cryptographic algorithms that supports, along with a random number Rc
- The server sends its certificate to C, a random number Rs, and the cipher that was chosen for implementing security services
- C verifies the certificate, extracts servers' public key
- C chooses a random number S (*pre-master secret*) from which the keys are going to be further computed, encrypts it with S' public key, and sends it along with the encryption of the keyed hash of the handshake messages
- This encryption of the keyed hash of the handshake messages can be formulated like follows (using the notation introduced later on in this book):

E(*h*(msgs, CLNT/client finished, *K*)), where *K* is the master secret generated from the pre-master secret S and the random numbers Rc and Rs. The client C sends a hash of the master key *K* and the handshake messages in order to prove that knows the key and to ensure that the tampering of the handshake messages would be detected. Kaufman et al. (2002) mentioned that although not necessary, the message digest is encrypted and integrity protected. The keys used for encrypting the keyed hash that is derived from hashing *K*, Rc and Rs.
The keys used for transmission are named *write keys*, while the keys used for reception are called *read keys.*

- For ensuring that the previous messages were not tampered in traffic and for proving that it knows the session keys, the server sends a keyed hash of all the handshake messages, encrypted with the write-encryption key and protected with the integrity-protection key (Kaufman et al., 2002)

E(*h*(msgs, SRVR/server finished, *K*))

For ensuring that the keyed hash sent by the client is different from the one sent by the server, the parties include a constant ASCII string value in the hash. In SSL 3.0 the initiator constant is CLNT and in TLS is *client finished*; the reply value is SRVR in SSL 3.0, respectively, *server finished* in TLS.

In this basic presentation of SSL/TLS protocol, just the server is authenticated, although authentication can be mutual if the client has also a certificate. Kaufman et al. (2002) made the observation that if the server wants to authenticate the client, it usually happens with a weak authentication mechanism (e.g. name and password encrypted with the session keys).

SSL Architecture

Stallings (2005) underlined that SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol, but rather is composed of two layers of protocols. The protocols within SSL/TLS are illustrated below in the figure below:

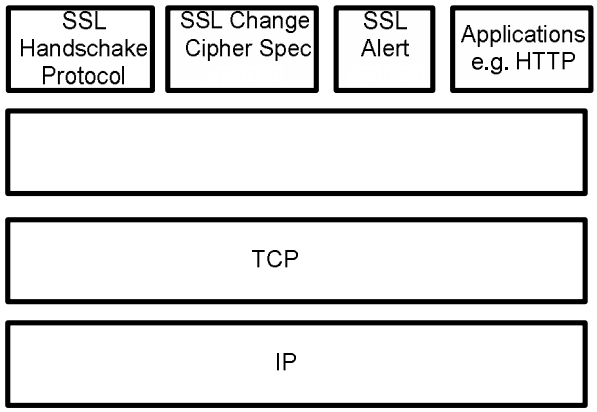


Figure 8: SSL/TLS Protocol Stack

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP) that provides the transfer service for Web client/server interaction can operate on top of SSL. Three other higher-layer protocols are defined as part of SSL: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. These SSL-specific protocols are used in the management of SSL exchanges.

SSL Record Protocol

Basically, the SSL Record Protocol applies the following operations on a transmitted message: fragments the data into manageable blocks, optionally the data is compressed, applies then a MAC, encrypts, adds a header, and transmits the result in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users.

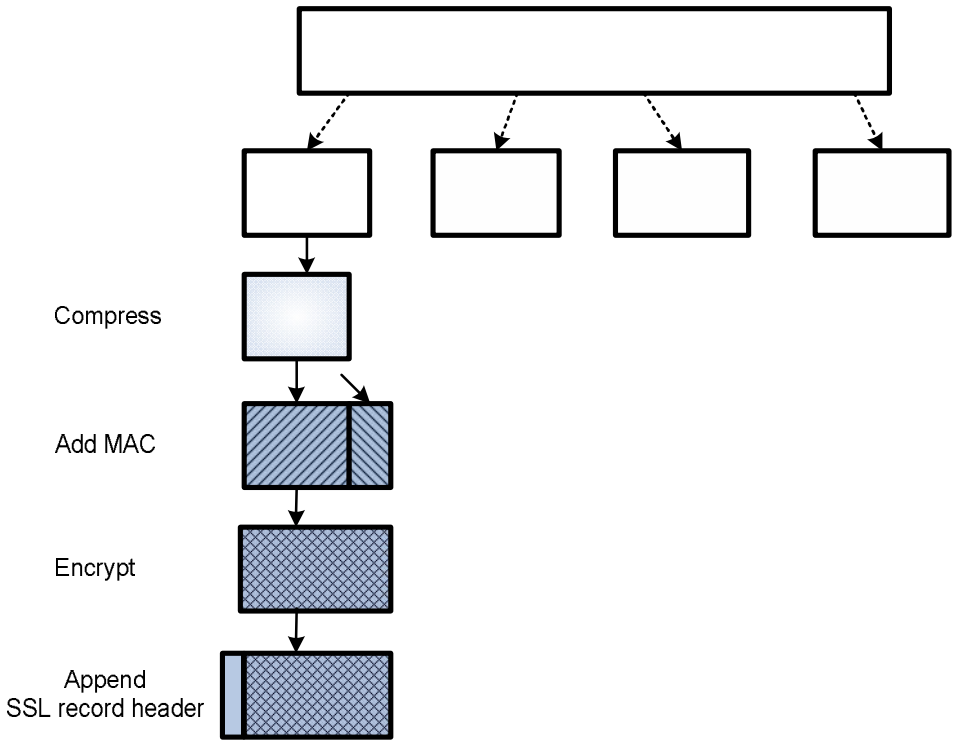


Figure 9: SSL Record Protocol Operations

Firstly, the SSL Record Protocol fragments the information blocks into SSLPlaintext records of 2^{14} bytes or less. A MAC is appended to each SSL record. It provides message origin authentication and data integrity services. Secure hash functions, such as MD5 or SHA-1, are used for MAC computations. The MAC is applied before encryption. In both SSL and TLS protocols, the MAC of the record also includes a sequence number, in order to detect missing, extra, or repeated messages, as well as replay attacks. Next, the compressed message plus the MAC are encrypted using symmetric encryption. Afterwards, if a block cipher is used for encryption, padding might be added after the MAC prior to encryption. The total size of the data (plaintext, MAC and padding) has to be a multiple of the block's length. Padding is added in the case the plaintext plus the MAC are not a multiple of block's length.

The encryption algorithms supported by SSL 3.0 are specified in the table below.

Block Cipher		**Stream Cipher**	
Algorithm	Key Size	Algorithm	Key Size
AES	128, 256	RC4-40	40
IDEA	128	RC4-128	128
RC2-40	40		
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

To summarize, the Record Protocol provides *connection security* that has two properties:

- *The connection is private*: Symmetric cryptography is used for data encryption. The keys for the symmetric encryption are generated uniquely for each connection and are based on a master secret negotiated by another protocol (SSL/TLS Handshake Protocol). It should be mentioned that the Record Protocol can also be used without encryption, but in the context of Jericho Project is highly advisable to encrypt the communications.

- *The connection is reliable*: Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g. SHA, MD5, etc.) are used for MAC computations. Although the Record Protocol can operate without a MAC, generally, when used as transport for negotiating security parameters, it is recommendable to use it with message integrity checks for transferring data in Jericho network.

Change Cipher Spec Protocol
The Change Cipher Spec Protocol is one of the three SSL-specific protocols and it is the simplest (Rhee, 2003; Stallings, 2005). This protocol signals transitions in ciphering strategies. It consists of a single message that is in fact a single byte with the value 1. The only purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

Alert Protocol
The Alert Protocol is being used to transmit SSL-related alerts to the peer entity via the SSL Record Protocol. An alert message consists of two parts, an alert level and an alert description. Also, alert messages are compressed and encrypted, as specified by the current state.

Handshake Protocol

The Handshake Protocol is considered to be the most complex part of SSL/TLS protocol. This protocol consists of a series of messages exchanged by the client and the server. Oppliger (2003) noted that the main aim of the Handshake Protocol is to have a client and server establish and maintain state information that is used to secure communications. The following operations occur in this protocol: the client and server agree on a common SSL protocol version, allows the server and client to authenticate each other, select the compression method and cipher spec, create a master secret from which the various session keys for message authentication and encryption may be derived. Next, we summarize an execution of the SSL Handshake Protocol between a client and a server in the figure below:

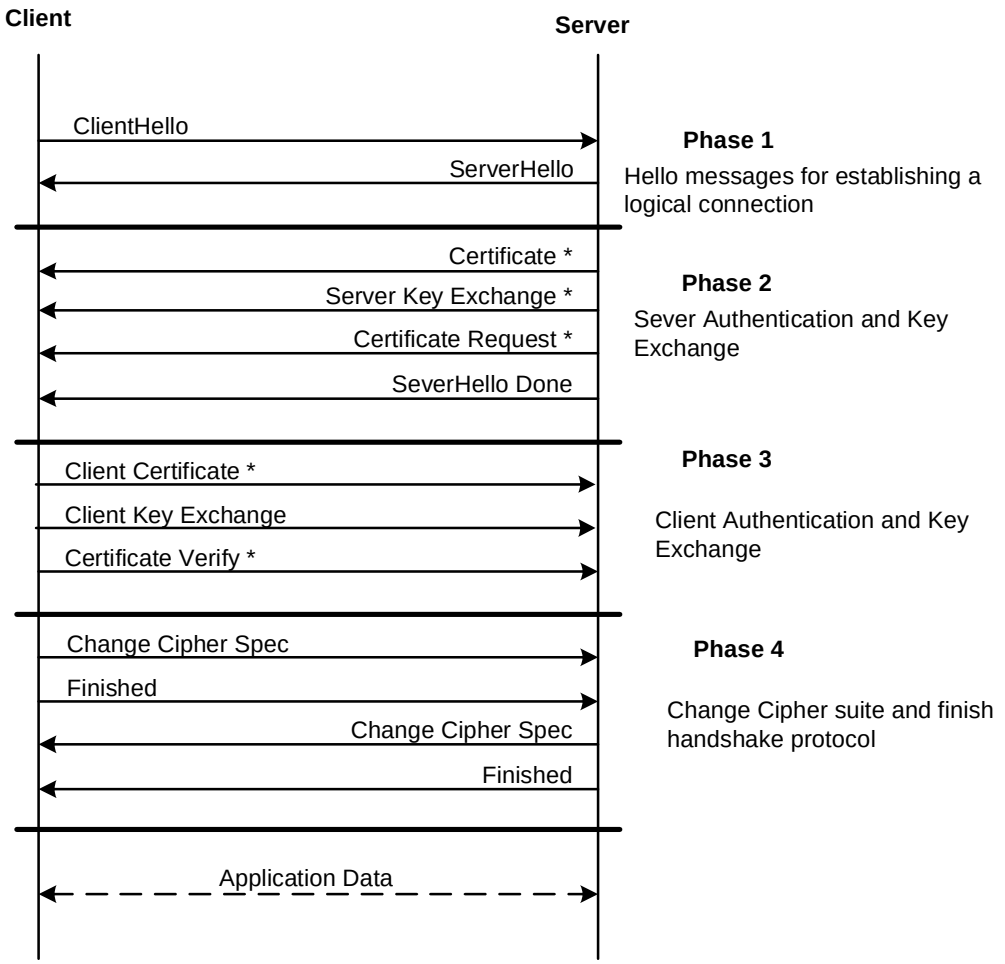


Figure 10: SSL Handshake Protocol

Asterisks (*) are optional or situation-dependent and messages that are not always sent

Phase 1
The aim of this phase in the Handshake Protocol is to initiate a logical connection ad to establish the security capabilities associated with it. The Client initiates the message exchange with a ClientHello message containing the following parameters:

Version: The highest SSL version understood by the client.
Random: A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.
Session ID: A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.
CipherSuite: This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.
Compression Method: This is a list of the compression methods the client supports.

Afterwards, the server sends the ServerHello message in response to the ClientHello message. This contains the same parameters mentioned above. In this case, the parameters have different values according to the selections made by the server. The Version field contains the lower of the version suggested by the client and the highest supported by the server. The Random field is generated by the server and is independent of the client's Random field. If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session. The CipherSuite field contains the single cipher suite selected by the server from those proposed by the client. The Compression field contains the compression method selected by the server from those proposed by the client.

The first element of the Cipher Suite parameter is the key exchange method that represents essentially the means by which the cryptographic keys for conventional encryption and MAC are exchanged.

The following key exchange methods are supported as specified in Internet Draft[1]for SSL 3.0:

- *RSA*: The secret key is encrypted with the receiver's RSA public key. A public-key certificate for the receiver's key must be made available.
- *Fixed Diffie-Hellman*: This is a Diffie-Hellman key exchange in which the server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key

[1] http://wp.netscape.com/eng/ssl3/draft302.txt

exchange message. This method results in a fixed secret key between two peers, based on the Diffie-Hellman calculation using the fixed public keys.

- *Ephemeral Diffie-Hellman*: This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.

- *Anonymous Diffie-Hellman*: The base Diffie-Hellman algorithm is used, with no authentication. That is, each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
- *Fortezza*: The technique defined for the Fortezza scheme. Stallings (2005) specified that Fortezza can be used in smart card encryption scheme.

After defining the key exchange method, CipherSpec follows and it includes the following fields:

- *CipherAlgorithm:* Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
- *MACAlgorithm:* MD5 or SHA-1
- *CipherType:* Stream or Block
- *IsExportable:* True or False
- *HashSize:* 0, 16 (for MD5), or 20 (for SHA-1) bytes
- *Key Material:* A sequence of bytes that contain data used in generating the write keys
- *IV Size:* The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

Phase 2

If the server uses a certificate-based authentication, then it sends its certificate to the client in a corresponding certificate message. The certificate must be appropriate for the selected cipher suite's key exchange algorithm, and is, generally, an X.509 certificate or a chain of certificates. Stallings (2005) noted that the certificate message is required for any agreed-on key exchange method except anonymous Diffie-Hellman. The client may send the same type of message for the server's certificate request message. The server's certificate request message includes two parameters: certificate type and certificate authority. The certificate type indicates the public key algorithm and its use (signature, authentication). The second parameter represents a list of the distinguished names of acceptable certificate authorities. It should be mentioned here that if fixed Diffie-Hellman is used, this certificate message functions as the server's key exchange message because it contains the server's public Diffie-Hellman parameters. Next, if it is needed, a server key exchange message may be sent. There are two situations when the server key exchange message is not required: when the server has sent a

certificate with fixed Diffie-Hellman parameters, or when it has been agreed that RSA key exchange will be used.

The server key exchange message is being sent in the following cases:

- *Anonymous Diffie-Hellman* : The message content consists of the two global Diffie-Hellman values (a prime number and a primitive root of that number) plus the server's public Diffie-Hellman key
- *Ephemeral Diffie-Hellman:* The message content includes the three Diffie-Hellman parameters provided for anonymous Diffie-Hellman, plus a signature of those parameters.
- *RSA key exchange, in which the server is using RSA but has a signature-only RSA key*: The client cannot simply send a secret key encrypted with the server's public key. Thus, the server must create a temporary RSA public/private key pair and use the server key exchange message to send the public key. The message content includes the two parameters of the temporary RSA public key (exponent and modulus) plus a signature of those parameters.
- *Fortezza*: The parameters of Fortezza are sent

Thus, a *server key exchange* message is sent in the case when the server has no certificate, or when the certificate is used for signature only (DSS or signing-only RSA certificates). Stallings (2005) described in detail how digital signatures are used in SSL 3.0. The authored underlined that hash functions and digital signatures are used not only for the parameters of the cryptographic algorithms used for encryption (RSA or Diffie-Hellman), but also for the random nonces from the initial hello messages. This ensures protection against replay attacks and misrepresentation. Moreover, details about the computation of the digital signatures are provided. In the case of a DSS signature, the hash is performed using the SHA-1 algorithm. In the case of an RSA signature, both an MD5 and an SHA-1 hash are calculated, and the concatenation of the two hashes is encrypted with the server's private key.
The final message in Phase 2 is the server done message. This is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response. This message has no parameters.

Phase 3
Further, upon receipt of the server done message, the client should verify that the server provided a valid certificate and check also that the server hello parameters are acceptable. After validation, if all is satisfactory, the client sends one or more messages back to the server (Stallings, 2005). In the case that the server required a client certificate, the client will send a certificate message. If the client has no suitable certificate available, it will send a *no certificate alert*. However, if the client authentication is needed and required, the server will answer with a handshake failure in the case of a *no certificate alert.* In the Internet Draft[1] for SSL 3.0 it is mentioned client Diffie-Hellman certificates must match the server specified Diffie-Hellman parameters. Next, the client will send a *client key exchange* message. The content of

[1] http://wp.netscape.com/eng/ssl3/draft302.txt

the message depends on the type of key exchange, as follows (Stallings, 2005; El Aoufi, 2006):

- *RSA:* The client generates a 48-byte *pre-master secret* and encrypts with the public key from the server's certificate or temporary RSA key from a *server key exchange* message.
- *Ephemeral or Anonymous Diffie-Hellman:* The client's public Diffie-Hellman parameters are conveyed if they were not already included in the client certificate.
- *Fixed Diffie-Hellman:* The client's public Diffie-Hellman parameters were sent in a certificate message, so the content of this message is null.
- *Fortezza:* The client's Fortezza parameters are sent.

Moreover, the client will send a *certificate verify message* in order to provide explicit verification and protection of a client message. Stallings (2005) explained that this message is only sent following any client certificate that has signing capability (e.g. all certificates except those containing fixed Diffie-Hellman parameters). This message signs a hash code based on the preceding messages. Firstly, a hash will be computed on all Handshake Protocol messages, the master secret and also two pad messages that are used in the hash. Stallings (2005) specified that if the user's private key is DSS, then it is used to encrypt the SHA-1 hash. If the user's private key is RSA, it is used to encrypt the concatenation of the MD5 and SHA-1 hashes. Anyway, the purpose of this *certificate verify message* is to verify the client's ownership of the private key for the client certificate. In the case that an attacker would misuse the client's certificate, this would be unable to send this message.

Phase 4

This phase completes the setting up of a secure connection. Firstly, a *change cipher spec* message is sent by the client, and the client copies the pending CipherSpec into the current CipherSpec. Further, the client sends then immediately the *finished message* under the new algorithms, keys and secrets. On the other hand, the server sends in response its own *change cipher spec* message, transfers the pending CipherSpec to the current one, and then sends its *finished message* under the new CipherSpec. The *finished message* is always sent immediately by the client, and then by the server, after a corresponding *change cipher spec* message in order to verify that the key exchange and authentication processes were successful. The *finished message* represents a concatenation of two hash values over the shared *master secret,* all the handshake messages up to this message, and a code that identifies the sender (either the client or the server). Finally, after the verification of the received *finished messages* by each entity, the handshake is complete and the client and server may begin to exchange application layer data. Further, application data is carried by the Record Layer and is fragmented, compressed and encrypted based on the current connection state.

To summarize, the Handshake Protocol provides connection security that has three basic properties:

- *Authentication*: The peer's identity can be authenticated using the means offered by public-key cryptography. Although, the authentication is optional, is generally required at least for one of the peers and it is advisable to make it bidirectional, as a requirement for secure communications in Jericho Project.
- *The negotiation of a shared secret is secure*: the negotiated secret is unavailable to eavesdroppers, and for any *authenticated connection* the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.
- *The negotiation is reliable*: no attacker can modify the negotiation communication without being detected by the parties to the communication.

All these properties are required and desirable for secure communications in Jericho networks.

Cryptographic computations

In the design of this protocol, public-key algorithms are used in the handshake protocol to authenticate parties and to generate shared keys and secrets. For all the accepted public-key algorithms (RSA, Diffie-Hellman, Fortezza) included in the specification of the protocol for key exchange methods, the same algorithm is used to convert the *pre-master secret* into the *master secret*. In order to create the master secret, a *pre-master secret* is first exchanged between two parties and then the *master secret* is calculated from it. The shared *master secret* is a value of 48 bytes (384 bits) and is generated for a session by means of secure key exchange. The length of the *pre-master secret* depends on the key exchange method. There are two ways for the exchange of the *pre-master secret*:

- *RSA*: A 48-byte *pre-master secret* is generated by the client, encrypted with the server's public RSA key, and sent to the server. The server decrypts the ciphertext using its private key to recover the *pre-master secret*.
- *Diffie–Hellman*: The client and the server generate a Diffie–Hellman common key. This private key is used as the *pre-master secret* and is converted into the *master secret*.

The pre-master secret should be deleted from memory once the master secret has been computed. This prevents attackers or malicious software to steal it from the memory (Kaufman et al., 2002). Both sides now compute the master secret as follows:

```
master_secret =
  MD5(pre_master_secret + SHA('A' + pre_master_secret +
     ClientHello.random + ServerHello.random)) +
  MD5(pre_master_secret + SHA('BB' + pre_master_secret +
     ClientHello.random + ServerHello.random)) +
  MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
     ClientHello.random + ServerHello.random));
```

From the *master secret* there will be generated six keys that will be used in data transmission process by each side (for encryption, integrity and for the initial vectors (IV)). For each connection, the *master secret* is shuffled with the random nonces produced by the client and server in order to produce the six keys used further in the communication. Basically, the *master secret* is used to generate keys and secrets for encryption and MAC computations for secure data transfer.

TLS Protocol

In this section we underline the differences between TLS 1.1, TLS 1.2 and SSL 3.0 that are of interest in the context of this research for Jericho Project. Although there are slightly differences between SSL 3.0 and TLS 1.0, the protocol remains substantially the same and the goal is to produce an Internet standard version of SSL. Anyway, as mentioned in RFC 2246[1] the differences are significant enough that TLS 1.0 and SSL 3.0 do not interoperate (although TLS 1.0 does incorporate a mechanism by which a TLS implementation can turn back to SSL 3.0). One of the differences between SSL 3.0 and TLS 1.0 is that in TLS there is user another algorithm for the computations of MACs used for integrity checking of the transferred messages. TLS makes use of the HMAC algorithm defined in RFC 2104[2]. Moreover, TLS 1.0 utilizes a pseudo-random function (PRF) to expand secrets into blocks of data for the purposes of key generation or validation. TLS supports all of the alert codes defined in SSL 3.0 with the exception of *no_certificate* option. A number of additional codes are defined in TLS 1.0. There are several small differences between the Cipher Suites available under SSL 3.0 and under TLS 1.0. For the key exchange, TLS supports all of the key exchange techniques of SSL 3.0 with the exception of Fortezza. With regard to the symmetric encryption algorithms, TLS includes all of the symmetric encryption algorithms found in SSL 3.0, with the exception of Fortezza.TLS 1.0 defines the following certificate types to be requested in a *certificate_request* message: RSA_sign, DSS_sign, RSA_fixed_DH, and DSS_fixed_DH. These are all defined in SSL 3.0 as well. In addition, SSL 3.0 includes RSA_ephemeral_DH, DSS_ephemeral_DH, and Fortezza_kea. TLS does not include the

Fortezza scheme. In TLS 1.0 *certificate_verify* message, the MD5 and SHA-1 hashes are calculated only over *handshake_messages*. In SSL 3.0 the hash calculations also included the *master secret* and pads. But because it was considered that these extra fields add no additional security, in TLS 1.0 it has been changed the calculation mode of the *certificate_verify* message. Moreover, in TLS 1.0 the calculation of the *master secret* and of the keying material used further in securely transmitting the data are

[1] http://www.ietf.org/rfc/rfc2246.txt
[2] www.ietf.org/rfc/rfc2104.txt

calculated also based on the *pre-master secret* and the random numbers of the entities, but the generating algorithms are slightly changed. Another difference refers to padding added prior to encryption of user data. In SSL 3.0 the padding added is the minimum amount required so that the total size of the data to be encrypted is a multiple of the cipher's block length. However, in TLS 1.0, the padding can be any amount that results in a total that is a multiple of the cipher's block length, up to a maximum of 255 bytes. Stallings (2005) pointed out that a variable padding length may be used to frustrate attacks based on an analysis of the lengths of exchanged messages. TLS 1.1[1] is intended to offer some minor security improvements in comparison with TLS 1.0. TLS 1.2[2] is a newer version of TLS 1.1 protocol that was released in an Internet Draft in March 2007. TLS 1.2 is intended to solve expected problems with digest algorithms from previous versions, has improved flexibility, especially for negotiation of cryptographic algorithms. The major changes in TLS 1.2 are related to the use of hash functions within the different operations in the protocol. Moreover, this new version of TLS protocol has an extra feature, namely support for authenticated encryption with additional data modes (AEAD). This is a symmetric encryption algorithm that simultaneously provides confidentiality and message integrity. In AEAD encryption, the plaintext is simultaneously encrypted and integrity protected. The input may be of any length and the output is generally larger than the input in order to accommodate the integrity check value.

Challenges of SSL/TLS

In theory is stated that SSL/TLS implementations (in particular servers) will be able to work with implementations of newer protocol versions. In reality, many SSL/TLS server implementations are broken with respect to forward compatibility: e.g. severs using different versions of the protocol are not able to establish communication or have implementation problems regarding the correct negotiation for the protocol versions to be used for secure communications. With regard to the transition from TLS 1.0 → TLS 1.1 → TLS 1.2, many servers do not accept ClientHello messages from TLS 1.1 clients. In such cases, various situations occur:

- No response from the server
- Connection closed immediately, with or without error
- Some refused to accept TLS 1.1
- Falls back to SSL v3, even if TLS 1.0 is supported.

However, it is not known whether or not this is caused by server implementation errors, or firewall rules. So, there are a series of problems with the compatibility between the different versions of the protocol, with the incorrect negotiation of the protocol version used by entities, and it seems that these problems would continue with the transition to TLS 1.2. Consequently, the TLS Working Groups should design specifications and security policies intended to solve the problems caused by non-compliant implementations of the protocol.

Security of SSL/TLS

[1] http://www.ietf.org/rfc/rfc4346.txt

[2] http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc4346-bis-03.txt

- SSL does not offer protection against traffic analysis attacks

Oppliger (2002) pointed out that by examining the unencrypted source and destination IP addresses and TCP port numbers, or examining the volume of transmitted data, a traffic analyst can still determine what parties are interacting, what types of services are being used, and sometimes even recover information about business or personal relationships.

- SSL does not protect against attacks directed against the TCP implementation, such as TCP SYN flooding or session hijacking attacks
- Phishing attacks
- Man-in-the-middle attacks

Northrup & Thomas (2004) pointed out that the SSL certificates help reduce the risk of attacks against Domain Name System (DNS). We add in this case, that the correct use and validation of the SSL certificates can reduce the occurrence of these risks. If attackers conduct a phishing attack for misleading the users to conduct transactions via a fake website, then they can collect any sensitive information about the users and their credentials with the man-in-the-middle attack. However, the Web sites have a certificate that is checked by the user. The fake (rogue) site could have a certificate containing the name of the impersonated organization but issued by an untrusted authority. Generally, the trusted certification authorities check thoroughly via the Registration Authorities the credentials of the entities to who they issue certificates. In this case the fake Web site has a certificate issued by an untrusted CA, the user will be prompted with an alert message being informed that the CA is untrusted. If users ignore these kind of alert messages and accept anyway the provided certificate, then they are exposed to a man-in-the-middle attack.

Certificate Requirements for SSL/TLS

One of the requirements of Jericho Project for secure communications refers to the authentication of entities involved in the data transfer. In Jericho Project, the ideal scenario regarding secure communications is to encrypt almost all the transmitted, choosing the encrypting ciphers according to the sensitive level assigned to data. This solution should hinder even the internal administrators from interception and/or reading the data. SSL/TLS seems a valuable tool for achieving the goals of secure communications in Jericho networks. At the moment, the adequate solution for end-to-end encryption with SSL/TLS would be the use of digital certificates. Digital certificates not only authenticate the entities, but also allow the use of encryption as a result of the public keys contained in the certificates. Thus, in order to conduct secure communications across Internet as required in Jericho Project, a mechanism is needed in order to validate and verify the identity of the entities that want to communicate. Moreover, this security mechanism would ideally allow you to encrypt and sign content also – these are other security services desired for secure communications in Jericho networks. Raina (2003) proposed an interesting model for the general use of the certificates. So, SSL/TLS certificates can be used for authentication and for end-to-end encryption as well. Moreover, an organization could be identified and authenticated with a valid “organizational” certificate. So, this could enable any entity that is being part of the organization to use that certificate for authentication. In this way, it would become easier to authenticate both the client and the server in an SSL/TLS session, and this would allow the deployment of SSL/TLS protocol at a larges scale for any kind of

transactions and data transfers. In SSL/TLS protocol a public key infrastructure (PKI) is the mechanism that provides the services and components to validate the identity of entities and exchange the keying material used further in the protocol. In order to be able to use SSL/TLS protocol, the server must have a suitable public-key certificate. Moreover, in the context of Jericho Project we recommend that also the clients should have and use public-key certificates. In essence, the client is configured with public keys of various "trusted organizations" (Certification Authorities (CAs)). The user operating at the client machine can modify the list, by adding or deleting keys. If the certificate sent by the server is signed by one of the CAs on client's list, after validating the certificate (e.g. expiration date, revocation), the client accepts the certificate. If the sent certificate is signed by another organization that is not on the client's list, the user will be prompted a pop-up window informing (see Figure 11 Security Alert) about this and will be asked if he/she wants to look at the certificate and/or import the signing authority into the trusted list. In this case, the user should definitely look at the certificate and check if possible the signing authority, and not add automatically this authority to the trusted list of authority that the client supports.

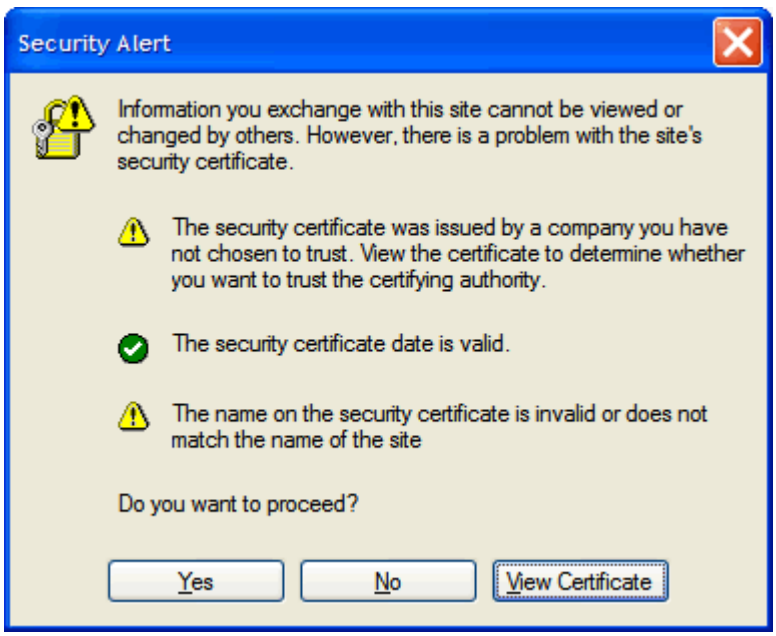


Figure 11: Security Alert for untrusted certificates

However, this warning does not prevent the user from establishing an SSL/TLS-encrypted session with the server for transferring data. But, the warning might cause the user to cancel the connection. Although establishing a connection to a server with an untrusted CA still provides encryption and message integrity. Using an SSL/TLS certificate issued by an untrusted CA defeats the purpose of the authentication provided by SSL/TLS protocol. In fact, these situations make the protocol vulnerable to man-in-the-middle attacks and phishing attacks. This is why, the users in Jericho networks should be aware by the risks posed by certificates signed by untrusted CAs. These certificates should be rejected and the SSL/TLS connection cancelled, until a solid verification of the issuing CA can be performed. The entities should perform certificate validation for the certificates sent by the parties with whom they communicate. The following tests should be performed for certificate validation:

- Ensure that the certificate chains to a trusted root certification authority (CA)
- Ensure that the certificate is time-valid
- Ensure that the certificate has not been revoked
- Ensures that the Domain Name System (DNS) name in the certificate's subject matches the DNS name in the HTTPS URL.

If the server' certificate passes all the validation tests, the browser extracts the certificate's associated public key used to transmit further the pre-master secret to the server. Depending on the cryptographic service provider (CSP) installed at the server, a Diffie-Hellman or a RSA negotiation allows the server and client to use the pre-master key to generate a symmetric session key using the same symmetric encryption algorithm. When the server wants to authenticate the client, it will send a request authentication to the client as well. SSL certificates can only be trusted if the root Certification Authority (CA) is trusted.

Certificates play an essential role in SSL/TLS protocol:

- An SSL Certificate enables encryption of sensitive information during online transactions
- Each SSL Certificate contains unique, authenticated information about the certificate owner
- Every SSL Certificate is issued by a Certificate Authority that verifies the identity of the certificate owner

Every SSL Certificate is created for a particular server in a specific domain for a verified business entity. The certificates are issued by a trusted authority, the Certificate Authority (CA). When the SSL handshake occurs, the browser requires authentication from the server. As we have already mentioned, in Jericho Project we recommend that the authentication of the client should be also mandatory, in order to enforce an adequate protection of the transferred data. Certificate-based authentication of the client is not required for SSL/TLS connections, but it definitely increases the security of the user's credentials. For an efficient and secure utilization of SSL/TLS protocol, the entities have to check each others certificates. If the information does not match or the certificates are expired, error messages or warnings are displayed. In the model of Jericho Project, there is another entity, the Trust Broker that should manage the certificates issued to the entities, establish trust relationships with Certification Authorities.

A possible scenario in the context of Jericho Project would be that the Trust Brokers issue certificates to clients and servers, like in the model of PGP in which there are created webs of trust. Trust Broker is intended to define and to manage rules for trust relationships between different organizations, and also to deal with cross-certifying hierarchies among companies (hierarchies of trust among companies). Another scenario would be that in which the organizations issue self-signed certificates. However, the way the Trust Broker manages or creates the certificates for the entities in Jericho networks is out of the scope of this book. The research topic of the Trust Broker in Jericho networks is dealt with by another member of Jericho Project research team.

Choosing a certificate provider

A recent market share report regarding the certificate providers was issued by Security Space[1] in June 2007. This report focuses on issuers of SSL enabling certificates found on web servers in April and May 2007.

Market share for certificates issuers:

Issuer	**Market Share**
VeriSign	23.6%
Equifax (Geotrust)	21.68%
Thawte	13.44%
Comodo Limited	8.5%
Starfield Technologies, Inc.	4.42%
DigiCert	2.38%
GoDaddy	1.4%
Entrust	1.18%
Network Solutions	1.17%

Nowadays, trust and privacy when transferring sensitive data over the Internet are of interest for every organization that conducts business online or establishes collaborations over the Internet. As we mentioned in the introductory part of this chapter, there are a series of requirements in Jericho networks for protecting the information against alterations, fraud, identity theft, eavesdropping etc. Basically, these problems can be addressed at this moment with an existing solution, namely the digital certificates issued by Certification Authorities. But next, there comes another question regarding how to choose a CA to provide certificates for authenticating the entities in an organizational and collaborative environment over the Internet. Essentially, all the CAs that issue certificates use the same technologies for producing and managing their certificates. The CAs offer also “site seals” that are clickable images that can be placed for instance on web pages in order to create trust.

[1] http://www.securityspace.com/s_survey/data/man.200705/casurvey.html?ca=VmVyaXNpZ24= accessed June 2007

Site Seals of different certificates providers:

The CAs differ generally in the ways they authenticate and verify the identity of entities that require digital certificates. The certificates providers should authenticate and verify strictly the information provided by entities requesting certificates, in order to be able to embed that information within the digital certificate, which in turn can be viewed, verified and confirmed by other entities. When an organization chooses to use SSL/TLS for securing the communications for different applications (HTTP, email, etc.), it has to determine from which certificate provider will obtain certificates for the clients and for the servers. This decision should be considered by the organizations in Jericho networks as well. In the context of Jericho networks, the Trust Broker plays an essential role. One of the roles of the Trust Broker can be to act as a Certification Authority, and, consequently, issue, manage, revoke certificates for entities. The Trust Broker can issue certificates for the clients and servers of the organization that implements and manages this entity (the Trust Broker), as well for partners organizations between which the Trust Broker has established trust relationships. There can be issued certificates for Web servers, for Web sites, for clients and for users. For instance, a Trust Broker can issue one-to-one certificates for authenticating users from other organizations (maps distinct certificates for each user/user account), or it can issue a many-to-one certificate for users that trust, but from other partner organizations. In the latter case, the individual users cannot be differentiated when connecting to a Web site or to a Web server, apart from the user accounts defined in the many-to-one mapping. Additionally, the Trust Broker would define also the capabilities of a certificate: for digital signatures, authentication, and encryption. Another scenario in Jericho networks would be that the Trust Broker could act as a Registration Authority that validates the credentials of the entities that submit requirements for certificates. In this way, the Trust Broker can manage directly the issuing of the certificates to entities. However, this research topic is out of the scope of this book and should constitute a separate research topic within Jericho Project. By using one of the models where the certificates are issued by the Trust Broker, or the Trust Broker plays the role of a Registration Authority, or the Trust Broker designs and manages a Circle of Trust[1] (CoT) for the issuers of the accepted certificates, an organization can enforce its security policies and certificate policies, and

[1] Idem 1

complies also with the guidelines and policies defined by the Trust Broker. There are different functional models for the Trust Broker[1] (collaborative model, consortium model, centralized model). In these models for the Trust Broker, there could be created CoT also for accepting the authentication of the entities with certificates issued only by the Trust Broker that manages the respective CoT or issued by another trusted Trust Broker of CA. This Circle of Trust would design in fact a Certificate Trust List (CTR), a concept already existing and being implemented by the organizations that use certificates. However, an organization can still purchase certificates for its servers from a CA until the framework of Trust Broker will be at a global scale implemented. A server can have also more certificates, so it can provide a certificate from a CA and/or from a Trust Broker. Anyway, any of these certificates has to provide transaction liability insurance for electronic transactions, data transmissions. Moreover, the certificates issued by the Trust broker can contribute to the establishment of trust relationships between different entities, to the design of a reputation concept based on the behaviour of entities using the certificates. Managing certificates can be complicated when a server has multiple certificates.SSL certificates can be used, for instance, to verify the identity of a Web site and to encrypt traffic sent between the client and the Web site. In this case, the SSL certificate identifies a Web site, and not a Web server. A single Web server can host multiple Web sites. Alternatively, a single Web site can be hosted on multiple Web servers to provide redundancy and scalability. If many websites of different entities are hosted on the same server, then the Web server needs different certificates for each Web site, in order to allow the verification of identity and encryption for each Web site. On the other hand, if a Web site is stored on different servers (e.g. copies of the same site) in order to allow the Web site to remain online in the event of a hardware failure, the same certificate of the Web site can be installed on all the servers. As we previously specified, clients can be also authenticated for secure communications and we recommend this in the context of Jericho Project. Authenticating clients can also lead to the enforcement of organization's security policy. For instance, rather than typing credentials or simply being connected to a Web site anonymously, entities can or can be enforced to use certificates for authentication. But, providing the certificate is not enough for the authentication of an entity. Besides this, an entity presenting the certificate must have also access the certificate's private key (the certificate is a public document that can be accessed by any entities) and also the issuer of the certificate should be included in the CTR of the entities with whom it communicates in order to be accepted as a valid means of authentication. Possession of the private key proves that the respective entity is the certificate's subject (Komar et al., 2004).Till this point we discussed mainly SSL certificates for Web servers and for the clients (namely, for HTTP protocol). However, SSL certificates can be used to protect several other protocols: LDAP, SMTP, POP3, NNTP, and SQL. So, SSL certificates can be used to encrypt LDAP and global catalog queries, database queries, to encrypt and protect messaging communications (Northrup & Thomas, 2004). When using SSL/TLS for these applications in order to achieve secure communications, the servers and the clients as well should also have certificates that are valid and are signed by trusted issuers (CAs or Trust Broker). The configurations for using SSL/TLS for these protocols and applications are out of the scope of this book.

[1] Adriaan Bruning, Trust Broker Services, Capgemini, 2008

Conclusions

SSL/TLS is an open standard that is widely deployed and supported by a variety of servers and clients. This means that SSL/TLS meets another requirement proposed by Jericho Forum, namely to adopt open, inherently secure standards.

Northrup & Thomas (2004) acknowledged that because SSL/TLS has been widely adopted, the security community has carefully examined the SSL/TLS standards, as well as their implementations. Due to this close examination, combined with the relative maturity of the SSL/TLS standards, the protocol has resulted in a highly secure method for authenticating clients and servers and protecting the privacy and integrity of communications.

According to VeriSign[1], SSL/TLS protocol should be deployed in the following cases:

- For (financial) transactions in electronic commerce
- For business partners accessing confidential information of other companies
- For companies that process sensitive or personal information such as addresses, birth dates, bank accounts etc.
- For complying with privacy and security requirements of the business partners
- For inspiring trust

All these cases are in fact scenarios that reflect the security requirements of Jericho Project for secure communications.

However, SSL/TLS protocol should be implemented or enabled only for the cases that there is a need to secure the communications. This is why, for instance, the data has to be classified[2] according to multiple security levels.

As Northrup & Thomas (2004) noted, although certificates are issued, for instance, to individual Web sites, SSL/TLS can be configured to offer protection only for sensitive data that has assigned higher levels of security (confidential, secret etc.) depending on the classification type used. One part of the Web site might require transmissions of encrypted data with SSL/TLS (by specifying HTTPS in the URL), and another part of the Web site might allow transmissions with data in clear (by specifying the simple form of the protocol, HTTP, in the URL).

The fact that the settings for the use of SSL/TLS can be configured for encrypting only parts of transferred data offers flexibility in security configuration. This allows providing end-to-end encryption of confidential data when necessary, and, at the same time, keeping the balance between security offered and performance[3] of the servers. (e.g. E-commerce sites typically use HTTPS only when exchanging private information, because this reduces limits on efficiency and performance due to the use of cryptography in securing the communications.)

An essential issue with reference to the implementation and the use of SSL/TLS, regards the fact that there should be ensured that the clients trust the root CA certificate of the Web Server's certificate chain[4]. If the Web Server certificate chains to an

[1] www.verisign.com

[2] Remco van Marle, Data classification & Information Leakage in Jericho Project, Capgemini, 2007

[3] The use of cryptographic mechanisms influences the performance of the data transmissions and of the Web servers

[4] A certificate chain is a sequence of certificates, where each certificate in the chain is signed by the subsequent certificate.

untrusted root CA or Trust Broker, users are warned that the certificate is not trusted, and the users should not connect to the respective server.

The correct validation and checking of the servers' certificates by the users offers protection against phishing attacks, and as well against man-in-the-middle attacks.

In fact, the phishing attacks lead to leakage of information and eavesdropping because the users are not aware of the risks of accepting invalid or fake certificates issued by untrusted CAs to attackers. Basically, these certificates issued by untrusted CAs to untrusted entities have no authentication value in a secure communications.

But there have been reported cases when certificates have been issued by reasonably trusted CAs to some attackers (e.g. fake organizations etc.) that conducted phishing attacks and misled the clients with their certificates.

IPsec

IPsec is a suite of protocols that operates at the network layer (layer 3 in the OSI model) for securing Internet Protocol (IP) network communications through the use of cryptographic security services that are independent of the interacting applications on the two hosts. IPsec provides end-to-end security for communications in different network configurations (e.g. client-to-client, client-to-server, and server-to-server) (Snader, 2005; Stallings, 2005; Stamp, 2006). IPsec offers network-level data integrity, data confidentiality, data origin authentication, and replay protection. Because IPsec is integrated at the Internet layer, it provides security for almost all protocols in the TCP/IP suite. Moreover, due to the fact that IPsec is applied transparently to applications, there is no need to configure separate security for each application that uses TCP/IP. Thus, by implementing security at the IP level, an organization can ensure secure networking not only for applications that include security mechanisms, but also for the security ignorant applications. However, Stamp (2006) confirmed that the major drawback of IPsec is that it is a complex protocol, characterized as "over-engineered." This makes the implementation of the protocol challenging. Due to the security services for securing the communications and its defining features, this protocol is further investigated in the context of Jericho Project.

Presentation

This security protocols consists of various cryptographic algorithms, security protocols, and key management protocols for achieving the features of secure communications mentioned at the beginning of this chapter. It ensures that the communications are secured from the source until they reach the destination, and its major advantage is that it is transparent to applications. By implementing security at the IP level, an organization can ensure secure networking and secure communications for all the applications. Due to the security services for securing the communications, this protocol presents interest for being investigates in the context of Jericho Project. IPsec protocols were first defined by RFCs 1825–1829, published in 1995. Then, in 1998, other RFCs were written and published, namely, RFCs 2401–2412. RFCs 2401–2412 are not compatible with 1825–1829, although they are conceptually identical. In 2005, third-generation documents, RFCs 4301–4309, were published. These third-generation documents standardized the abbreviation of IPsec to uppercase "IP" and lowercase "sec". Support for the features presented in these RFCs is mandatory for IPv6 and optional for IPv4. The security features are implemented as extension headers that follow the main IP

header. The extension header for authentication is named the Authentication header. The one for encryption is known as the Encapsulating Security Payload (ESP) header (Stallings, 2005).

Essentially, IPsec consists of three major protocols (Snader, 2005):

- *Authentication Header (AH)*: This protocol provides data origin authentication, data integrity, and replay protection
- *Encapsulating Security Payload (ESP)*: This protocol provides the same services as AH but also offers data privacy (confidentiality) through the use of encryption
- *Internet Key Exchange (IKE)*: This protocol provides all important key-management functions. The alternative to IKE is manual keying, supported by IPsec as well.

These protocols can be combined and configured in a flexible manner, but this leads to an increased overall complexity of the protocol. As Snader (2005) mentioned as a general rule, complexity is the enemy of security, so the increase in complexity can lead to a decrease in security. Ferguson & Schneier (1999) stated as well that security's worst enemy is complexity. Consequently, the increase in complexity leads to the fact that, typically, IPsec is more difficult to configure and manage.

IPsec Properties

IPsec protocol comprises two protocols that are employed to provide security of communications, Authentication Header (AH) and Encapsulating Security Payload (ESP). IPsec enables a system to select the necessary security protocols, decide the algorithms and primitives to use further for achieving the desired security services. IPsec provides the following security features (Poddar et al., 2003; Stallings, 2005; Pujolle, 2007):

- *Data source authentication*: Ensures that the communication takes place with a client that is authenticated and authorized for communication
- *Access control*: Ensures that the communication occur with a client that is IPsec enabled.
- *Integrity*: Assures that the received data packets are identical with the data packets sent by the data source. Also assures that the data packets have not been altered.
- *Anti-replay protection*: Verifies that no redundant data packets are received
- *Confidentiality*: Enables encryption of transmitted data, so, that the data remains confidential in traffic and protection is offered against eavesdroppers. It offers also the possibility to encrypt the IP packet header
- *Key management*: Offers secure exchange of keys

Stallings (2005) investigated the security services provided by AH and ESP protocols. AH and ESP protocols represent the core IPsec protocols for communicating data securely. ESP protocol can be used with or without an authentication option. Both AH and ESP protocols ensure access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

IPsec services (after Stallings, 2005):

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	√	√	√
Connectionless integrity	√		√
Data origin authentication	√		√
Rejection of replayed packets	√	√	√
Confidentiality		√	√
Limited traffic flow confidentiality		√	√

It is obvious that some of the functionalities of AH (e.g. data integrity and authenticity) and ESP (e.g. confidentiality, data integrity and authentication) protocols overlap. Olifer & Olifer (2006) pointed out that the distribution of security functions between AH and ESP is justified by limiting export and/or import to the encryption tools, a practice adopted by many countries. Thus, these protocols can be used independently or together for achieving the necessary features for secure communications. For instance, when no encryption is allowed due to imposed limitations, it is possible to supply the system only with the protection provided by AH protocol. Obviously, according to the requirements for secure communications in Jericho networks, the protection of transmitted data offered by AH protocol only won't suffice. When AH protocol is used, the receiving party can only check whether the data was sent by the node from which it was expected (data origin authentication) and whether it is in the same form it was sent (integrity checking). Thus, for achieving confidentiality of the transferred data, it is necessary to use the ESP protocol. Basically, IPsec encompasses three functional areas: authentication, confidentiality, and key management (Stallings, 2005). Based on the requirements defined for secure communication in the context of Jericho Project, we investigate further IPsec protocol as a potential solution for secure communications over the Internet. IPsec is below the transport layer (TCP, UDP), thus it is transparent to applications. IPsec can be implemented in the firewall or router, as well on clients. In the context of Jericho Project, we recommend based on Jericho Forum Commandments that IPsec should be implemented on clients, thus these become IPsec enabled. When IPsec is implemented in end systems, upper-layer software, including applications, is not affected. Thus the protocol offers a transparent method of assuring end-to-end security of the communications over the Internet. Another feature of IPsec is that it can be transparent to users. Stallings (2005) mentioned that there is no need in training users on security mechanisms of issues regarding the use of cryptographic keys. Further we will present shortly the IPsec process of protecting a

transmitted message on an IP network (Poddar et al., 2003). This consists of the following steps:

- Firstly, a secure management connection is built for further negotiation
- A negotiation is established between two computers
- Negotiate what security key(s) to use for communication
- Exchange of the security keys (using key management protocols) in order to provide confidentiality for the transferred message
- The computers involved in the communication process negotiate about the encryption algorithms required to protect the message
- The Security Association (SA) messages facilitate the exchange of information about encryption algorithms
- The IPsec Domain of Interpretation (DOI) controls the IPsec process by defining message formats, exchange types, and conventions to refer security protocols, cryptographic algorithms, and security keys.
- Integrity check is performed to ensure that the data does not change during transmission
- IPsec provides anti-replay features for preventing the transmission of redundant data

Security Associations

Quiggle (2001), Rhee (2003), Stallings (2005), Pujolle (2007) etc. stated that a Security Association (SA) is a one–way connection/relationship between a sender and receiver that permits security services for the traffic carried on it. If a peer relationship is required, for two-way secure exchange and communication transfer protection, then there are required and defined two SAs. Thus, before establishing secure communications for data in traffic, an SA has to be established and shared between the communicating parties. Security services are afforded to an SA for the use of AH or ESP, but not both (Stallings, 2005). Basically, the SA messages support the exchange of information about the encryption algorithms that will further be used for securing the communications. Poddar et al. (2003) defined SA as an agreement between a sender and a receiver on a network in order to determine the security options. Olifer & Olifer (2006) specified that the procedure of establishing an SA starts with the mutual authentication of both parties. This ensures that the transmitted data is exchanged between authenticated parties. The SA parameters that are later chosen define which of the two protocols, AH or ESP, will be used for data protection and which functions will be carried out by the security protocol. There can be chosen only authentication and integrity, or it can also ensure confidentiality. Besides this, other important parameters of an SA are the private keys used by the AH and ESP protocols. A security association is uniquely identified by three parameters:

- *Security Parameter Index (SPI)*: This field is present in the AH and/or ESP headers. This parameter enables the receiving entity to select the SA that defines the parameters under which a packet will be processed.
- *IP destination address*: This represents the destination address of the IPsec peer, which may be an end user system or a network system such as a firewall or router.

- *Security Protocol Identifier*: This indicates whether an SA is an AH or ESP Security Association.

Consequently, in any IP packet (e.g. either an IPv4 datagram or an IPv6 packet), the Security Association is uniquely identified by the IP destination address in the IPv4 or IPv6 header and the SPI in the enclosed extension header (AH or ESP). Olifer & Olifer (2007) specified that in IPSec protocol exist possibilities for using both automatic and manual methods of establishing an SA. In the case of manual established SA, the administrator configures end nodes to ensure that they support association parameters. Otherwise, in the case of using an automated procedure of establishing an SA, IKE protocols operating on different sides of the channel choose parameters in the course of the negotiation process. Thus, IPsec is a flexible tool for configuring the parameters used and for securing the transfer of data. An SA established between two IPsec enabled entities defines the following parameters (Stallings, 2005):

- *Sequence number counter*: A 32-bit value used to generate the Sequence Number field in AH or ESP headers (required for all implementations)
- *Sequence Counter Overflow*: A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA (required for all implementations).
- *Anti-replay window*: Used to determine whether an inbound AH or ESP packet is a replay. If a packet has already been received or fails authentication, the packet is discarded and audit logs are generated (required for all implementations)
- *AH Information*: Authentication algorithm, keys, key lifetimes, and related parameters being used with AH (required for AH implementations)
- *ESP Information*: Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP (required for ESP implementations)
- *Lifetime of the respective Security Association*: A time interval or byte count that cannot be exceeded, and after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur (required for all implementations)
- *IPSec Protocol Mode*: Tunnel, transport, or wildcard (required for all implementations)
- *Path MTU*: Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables (required for all implementations)

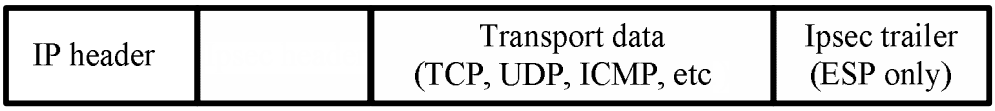
Quiggle (2001) indicated that in order to minimize the risk for any encrypted message to be broken by brute force attacks, Security Associations have a time limit that cannot be exceed. Otherwise, the IPsec enabled peers have to renegotiate again all the parameters used for secure communications. Moreover, some SAs implement message limits as well (e.g. after 10MB of data have been exchanged, the SA is established again and the parameters renegotiated).

Operation Modes

IPsec protocol can operate in two modes: *transport* and *tunnel* modes. Basically, AH and ESP protocols that are embedded in IPsec can protect data in two modes: transport and tunnel.

Transport mode provides protection primarily for upper-layer protocols (a TCP packet or UDP segment or an Internet Control Message Protocol (ICMP) packet). Stallings (2005) noted that typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). Snader (2005) acknowledged that transport mode is transport mode is meant to be used between two fixed hosts, namely when the end points are the final destinations of the transferred data. Thus, the hosts are IPsec enabled, and the communications are 'end-to-end' secured in traffic. In the case of a host running AH or ESP protocol over IPv4, the payload is the data that follows the IP header. For IPv6, the payload is the data that follows both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header that might be included in the protection. Rhee (2003), Stallings (2005) specified that ESP in transport mode encrypts and optionally authenticates the IP payload, but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header. Moreover, the authors noted that a transport mode SA provides security services only for higher-layer protocols, not for the IP header or any extension headers preceding the ESP header.

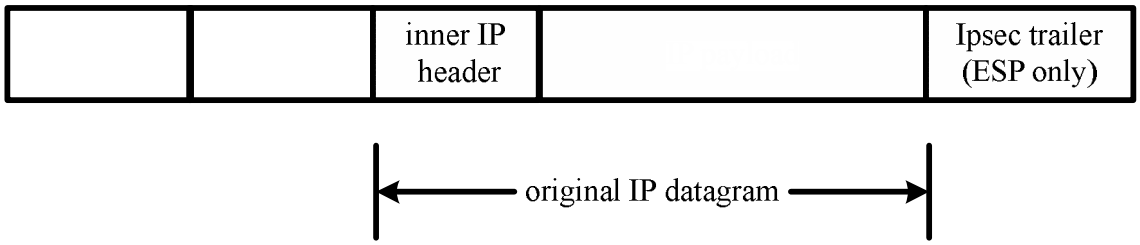
The transport mode of working:



When AH is used, then there is no IPsec trailer.

Basically, IPsec is used in transport mode for securing the host-to-host communications. So, IPsec in transport mode is used to encrypt and validate the integrity of communications between two computers. Therefore, IPsec can protect traffic between Web servers and database servers, or between Web clients and Web servers. For instance, when an IPsec client attempts to initiate a connection to an IPsec server, the client and server negotiate IPsec integrity and encryption protocols. After the IPsec connection is established, the data is transferred within the IPsec connection.

Tunnel mode provides protection to the entire IP packet. Shortly, in tunnel mode, the source data packet is encapsulated in the new IP packet and the data is transmitted over the network on the basis of the new IP packet header. So, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields are treated as the payload of new IP packet that will have a new IP header. The initial data packet travels through a "tunnel" from one point of an IP network to another, without being exposed to traffic analysis (the original packet is encapsulated and transmitted in a tunnel). Besides this, the new packet may have totally different source and destination addresses, in this way adding to the security. Tunnel Mode Encapsulation:



Typically, tunnel mode can be used when one or both ends of an SA are a security gateway (e.g. a firewall or a router) that implements IPsec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPSec. But, this contravenes with Jericho Forum Commandments[1] that, essentially, state that individual systems and data will need to be capable of protecting themselves. As Snader (2005) mentioned IPsec in tunnel mode provides end-to-end security. Snader (2005) added that the tunnel mode could be as well used for securing the traffic between two hosts In this case, the source and destination addresses of the inner and outer IP headers would be the same. The disadvantage of this manner of using the tunnel mode is the extra bandwidth required by the additional IP header. The author suggested that transport mode could be used and regarded as an optimization of tunnel mode for the special case of two fixed hosts.

Ferguson & Schneier (1999) pointed out that it would be easy to compress the data in the inner IP header (in the tunnelling mode). This would be a method to eliminate the transport mode and the extra complexity it entails, with no cost in extra bandwidth. Stallings (2005) summarized the transport and tunnel model functionality.

[1] www.opengroup.org/jericho/commandments_v1.2.pdf (Version 1.2 May 2007) accessed June 2007

.

Tunnel Mode and Transport Mode Functionality (after Stallings, 2005):

	Transport Mode SA	**Tunnel Mode SA**
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

In the context of Jericho Project, based on the requirements that we determined for secure communications, which are underpinned by Jericho Forum Commandments, IPsec should be used in transport mode for providing end-to-end encryption and security for the transmitted data. In essence, in Jericho Project, the network is the Internet and the communications occur over the Internet. Due to this aspect, the transport mode for securing host-to-host communications is recommended in Jericho Project.

Authentication Header (AH)

Authentication Header (AH) protocol provides authentication of IP packets, data integrity, and protection against a replay attack. But, AH does not comprise security mechanisms for achieving confidentiality. As we have already mentioned there are sometimes limitations regarding the use of cryptographic algorithms, and in those cases AH protocol can be used. Essentially, AH protocol adds a supplementary field to the IP packet that enables, upon receipt, the verification of the authenticity of the transferred data. The data integrity feature ensures that the modifications of a data packet in transit are detected, while the authentication feature enables an end system to authenticate the user or application and filter traffic in accordance with the defined policies. Stallings (2005) added that it also prevents the address spoofing attacks that occur nowadays in the communications over Internet. AH also secures against the replay attacks as well. The Authentication Header consists of the following fields:

- *Next Header (8 bits)*: Identifies the type of header immediately following this header (e.g. a TCP segment, an UDP message, or an ICMP packet). It represents the protocol number of the AH payload.
- *Payload Length (8 bits)*: This represents the length of Authentication Header in 32-bit words, minus 2.
- *Reserved (16 bits)*: For future use.

- *Security Parameters Index (32 bits)*: Identifies an SA.
- *Sequence Number (32 bits)*: Represents a counter that increases by 1 for each AH datagram that a host sends for a particular SA.
- *Authentication Data (variable)*: A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for the respective packet.

The Sequence Number field offers anti-replay service. So, this field is used for protecting the packet from being reproduced by attackers that may try to reuse the sniffed protected packets sent by the authenticated user. The sender sequentially increases the value of this field in each new packet transmitted within the framework of this SA, so the arrival of a duplicate will be noticed by the receiving party (the protection against false duplication has to be enabled within the respective SA (Olifer & Olifer, 2007). The Integrity Check Value (ICV) is a message authentication code or a truncated version of a code produced by a MAC algorithm. The current specification dictates that a compliant implementation must support the following security mechanisms: HMAC-MD5-96, HMAC-SHA-1-96. These security mechanisms use the HMAC algorithm, the first with the MD5 hash code and the second with the SHA-1 hash code[1]. In both cases, the full HMAC value is calculated but then truncated by using the first 96 bits, which is the default length for the Authentication Data field. In the context of Jericho Project, when chosen to implement IPsec with AH protocol, it is recommended to use the authentication and integrity services directly between two hosts (e.g. a server and a client), namely to use a transport mode SA. The use of a transport mode SA for IPsec enabled hosts, using AH protocol, provides end-to-end authentication in Jericho networks. To summarize, AH protocol in transport mode provides: authentication, data integrity, and anti-replay protection to the entire data packet (IP header and IP payload). Moreover, in Jericho enabled networks it is advisable to use AH protocol for the transfer of data that can has attached low security levels (e.g. public data), but still needs to be checked for integrity and authentication.

Encapsulating Security Protocol (ESP)

The Encapsulating Security Payload (ESP) provides as well authentication, data integrity, and anti-replay protection, and besides these, also confidentiality. ESP protocol consists of a header, the ESP header attached to the data packet in order to encrypt the data, and a trailer, the ESP trailer attached to the data packet in order to provide authentication. Rhee (2003) pointed out that data authentication and integrity are joint services offered as an option with confidentiality. The anti-replay service is chosen only if data origin authentication is selected. Moreover, the service is effective only if the receiver checks the sequence number.

ESP contains the following fields:

- *Security Parameters Index (32 bits)*: Identifies an SA for communication.
- *Sequence Number (32 bits)*: An increasing counter value that calculates a record of data packets sent over the SA, and provides anti-replay protection.

[1] We describe the security mechanisms for achieving security services that lead to secure communications in more details in Chapter 4 of this book.

- *Payload Data (variable*): Represents transmitted data protected by encryption. For the cryptographic algorithms that require an initialization vector (e.g. block cipher used in CBC mode), this is included in the payload data (Snader, 2005).
- *Padding (0-255 bytes)*: ensures that the encrypted data and the padding are no longer than 256 bytes.
- *Pad Length (8 bits)*: Indicates the pad length in bytes. The receiver uses this value for removing the padding bytes after decrypting the data.
- *Next Header (8 bits)*: Identifies the type of data contained in the payload data field by identifying the first header in that payload
- *Authentication Data (variable)*: A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV). The ICV is calculated over the entire ESP packet except for the authentication data field itself.

These fields are grouped in the ESP packet in four parts:

- *The ESP header*: contains the SPI and sequence number fields
- *The payload*: contains the payload data field
- *The ESP trailer*: contains the padding, pad length, and next header fields
- The ESP authentication data: contains the ICV

Encryption and Authentication Algorithms used in ESP

In ESP, confidentiality could be selected independent of all other services offered by this protocol. But, as Rhee (2003) suggested, the use of confidentiality without integrity and data origin authentication may be subject to active attacks that undermine the confidentiality service. Stallings (2005) specified that payload and the ESP trailer are encrypted by the ESP service. Based on the specifications of IPsec protocol, a number of other algorithms can be used for encryption. These[1] include: three-key 3DES, RC5, IDEA, three-key triple IDEA, CAST, and Blowfish. However, Olifer & Olifer (2007) affirmed that IPsec can employ any symmetric encryption algorithm for encrypting the data. But, we add that there should be used any symmetric-key cryptographic algorithm that has not been yet broken and has a sufficient key size for not being vulnerable in the near future. In RFC 4305[2] - "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", there are also listed the encryption algorithms for ESP.

[1] Some of these algorithms are described in Chapter 4 of the book

[2] http://www.ietf.org/rfc/rfc4305.txt accessed June 2007

Encryption algorithms recommended in RFC 4305:

Algorithm	Requirement	Note
NULL	MUST	ESP encryption and authentication are optional, so support for the two "NULL" algorithms is required to maintain consistency with the way these services are negotiated. Authentication and encryption can each be "NULL", but they MUST NOT both be "NULL".
TripleDES-CBC	MUST- he authors of RFC 4305 expect that at some point in the future this algorithm will no longer be a MUST.	RFC2451
AES-CBC with 128-bit keys	SHOULD+ The authors consider that it is likely that an algorithm marked as SHOULD+ will be promoted at some future time to be a MUST.	RFC3602
AES-CTR	SHOULD	RFC3686
DES-CBC	SHOULD NOT	RFC2405

The authentication algorithm employed for the ICV computation is specified by the SA. For ensuring data integrity and authentication in the case of communication between two points, suitable algorithms include Message Authentication Codes (MACs) based on symmetric encryption algorithms or on one-way hash function (e.g. MD5 or SHA-1). For multicast communication, one-way hash algorithms combined with asymmetric signature algorithms are appropriate. Basically, the data packet is authenticated by computing the ICV over the ESP header, payload, and ESP trailer fields, using the algorithm and key specified in the SA. Stallings (2005) and Rhee (2003) pointed out that compliant implementations must support HMAC-MD5-96 and HMAC-SHA-1-96 for providing integrity and authentication. In RFC 4305[1] there are also listed the recommended authentication algorithms for ESP.

[1] http://www.ietf.org/rfc/rfc4305.txt accessed June 2007

.

Authentication algorithms recommended in RFC 4305:

Algorithm	Requirement	Note
HMAC-SHA1-96	MUST	RFC2404
NULL	MUST	ESP encryption and authentication are optional, so support for the two "NULL" algorithms is required to maintain consistency with the way these services are negotiated. Authentication and encryption can each be "NULL", but they MUST NOT both be "NULL".
AES-XCBC-MAC-96	SHOULD+ the authors consider that it is ikely that an algorithm marked as SHOULD+ will be promoted at some future time to be a MUST.	RFC3566
HMAC-MD5-96	MAY Weaknesses have become apparent in MD5 (we specified this in Chapter 4).	RFC2403

However, although ESP provides data-origin authentication, it cannot authenticate all the data within the packet like AH protocol does. Quiggle (2001) recommended in the case when a higher level of security is required for the transferred data, to deploy both ESP and AH (this will decrease the overall performance of the transmission). If ESP is chosen to achieve both authentication and confidentiality, then encryption is performed first, and then the authentication is provided. Stallings (2005) pointed out that encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. The author explained also different combinations of SAs for achieving confidentiality and encryption in varied ways. However, the cryptographic primitives and algorithms used in the protocols comprised in IPsec for achieving different security services are extended by vendors with other algorithms. Nevertheless, when choosing which cryptographic algorithms and primitives will be applied in IPsec, there should be first considered the security requirements and objectives that have to be accomplished, and ultimately there should be taken into consideration the performance issues. Basically, the user can decide which security algorithm to use for an application depending on the nature of security to be provided.

Key Exchange

The key management part of IPsec comprises the determination and distribution of secret keys. Rhee (2003) mentioned that key establishment is at the heart of data

protection that relies on cryptography. Moreover, a secure key distribution for the Internet represents an essential part of data protection. Before establishing a secure session, the communicating parties need to negotiate the terms that are defined in the SA. IPsec supports two types of key management:
Manual: A system administrator manually configures each system with its own keys and with the keys of other communicating systems.
Automated: An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration. This can be done by means of an appropriate protocol. IPsec implements as automated management protocol the Internet Key Exchange Protocol (IKE).

The Internet Key Exchange (IKE) protocol handles the problem of key management by negotiating security associations between peers that want to communicate securely. Snader (2005) summarized the working of IKE: the peers perform a Diffie-Hellman exchange to obtain a shared secret that they use to generate keying material for the encryption and authentication algorithms used to protect the transmitted data. However, IKE must protect itself against denial-of-service attacks, replay attacks, man-in-the-middle attacks, and other attempts to subvert the secure exchange of keys. Besides exchanging keying material, IKE negotiates the encryption, authentication, and other cryptographic primitives used for achieving the goals of secure communications. As described in the literature (Poddar et al., 2003; Rhee, 2003; Snader, 2005; Stallings, 2005, Pujolle, 2007 etc.), IKE is a hybrid protocol based on other protocols: Internet Security Association and Key Management Protocol (ISAKMP), Oakley Key Determination protocol (Oakley) and SKEME protocol. IKE protocol has been designed for achieving the following goals within IPsec (Quiggle, 2001):

- Provides a means for parties that use IPsec to agree on the protocols, algorithms, and keys to be used for a key exchange
- Provides authentication of the IPsec enabled peers
- Manages the keys after they have been agreed upon

Besides negotiating the SAs and handling the key exchange, IKE authenticates each peer to the other. This ensures that each node can be sure of the identity of its peer. There are four ways to do this authentication: shared secrets, digital signatures, public key encryption of nonces, and revised public key encryption of nonces. IPsec provides certificate-based authentication as well, as a service for secure communications. Komar et al. (2004) specified that certificates can be used to authenticate the peers in an IPsec association. After the peers are authenticated, IPsec is used to encrypt and digitally sign all communications between the two endpoints. The certificates are used only for authentication purposes. For achieving all these functions, IKE combines the functions provided by ISAKMP and Oakley, and is referred in the literature as referred to as ISAKMP/Oakley protocol.

Oakley: This is a key exchange protocol based on the Diffie-Hellman algorithm that provides details for perfect forward secrecy for keys, identity protection, and authentication services.
ISAKMP: Is designed as a framework that expresses additional protocols for establishing security associations, for performing authentication and key exchanges.

ISAKMP is independent of any particular key-exchange method. In fact, it is a general framework that can support many key-management protocols.
Due to the space allocated for this book and the scope of research, we aim to provide a general perspective of the security protocols that can accomplish the requirements for secure communications in Jericho networks. Thus, we do not describe in detail the mechanisms employed by IKE in IPsec protocol. However, in the context of Jericho Project, we recommend as appropriate the automated key management system. In Jericho networks it is required to ensure the security of communications over the Internet, and this corresponds to the applicability of the automated key management.

Conclusions

IPsec protocol can be used to secure the communications over the Internet in a transparent way for the applications. As we have already mentioned, because IPSec operates at the network layer as an extension to the IP protocol, it provides end-to-end encryption, meaning that the source computer encrypts the data, and it is not decrypted until it reaches its final destination. Thus, in the context of Jericho Project, IPsec can be used to secure the communications of organizations over the Internet, including the communications regarding confidential data inside the organizations, as well with the partners. Besides these, it can also be used for enhancing electronic commerce security, even for some Web and electronic commerce applications that already have built-in security protocols (Stallings, 2005). Northrup & Thomas (2004) pointed out that IPsec can be also used to provide protection against:

- Network-based denial-of-service attacks from untrusted computers
- Data corruption
- Data theft
- User-credential theft

The principal security feature offered by IPsec that enables it to ensure the security of the communications in so many cases is that it can encrypt and/or authenticate all traffic at the IP level. Consequently, all distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured by using IPsec. So, in addition to the improved security, IPsec protocol can be used for enabling communications between remote offices and remote access clients across Internet.
Ferguson & Schneier (1999) concluded that even if there is a lot of criticism against IPsec because of various reasons (e.g. quality, services provided, complexity, security flaws etc.) it is still probably the best IP security protocol available. The main criticism of the authors for IPsec regards the complexity of the protocol. They stated that IPsec contains too many options and too much flexibility, and there are often several ways of doing the same or similar things within the protocol. However, planning and configuring an IPSec infrastructure is a complex task due to the inherent complexity of IPsec. In this chapter we presented the features of IPsec, the main aspects of the protocols that it contains, and the security services provided for secure communications. In conclusion, for Jericho Project it is recommendable to use IPsec in transport mode to secure the communications across Internet between two hosts. For using IPsec in transport mode to encrypt and authenticate the data transmitted over Internet, the clients have to be IPsec enabled. However, many computers nowadays are not IPsec enabled. As a result, computers that are IPsec enabled are typically configured to request peer computers to use IPsec to for improving the security of the connection. Northrup &

Thomas (2004) described how two hosts establish a connection. Thus, if two computers determine that they both have IPSec configured, and can agree upon a set of security standards, they can begin to use IPSec for securing the communications. This process is known as IPsec negotiation. If the computers fail to establish a negotiation (e.g. the computers are not IPsec enabled, or might not have the same security protocols enabled etc.) they might revert to unprotected IP communications or not communicate if they cannot use IPsec. Of course, it is preferable not to communicate at all when the negotiation fails and sensitive data needs to be securely transmitted over Internet. Bragg (2003) suggested that negotiation policies should be used in the following cases:
Computer authentication is required before a connection is allowed: Thus, connections from computers that cannot authenticate or are not IPsec enabled can be blocked.
Sensitive data must be transmitted: IPSec policies can be required to negotiate encryption type, and thus ensure all data is protected.
Ensure connection from specific computers only: Authentication can limit connections. Explicit filters can limit the connections to specific computers using their IP addresses.
When choosing to implement IPsec, there should be selected the option that best meet the authentication and encryption needs. It is recommendable to choose both data origin authentication and encryption features for protecting the transmitted data. The encryption algorithms can be chosen from the one specified in the RFCs for IPsec or might be also recommended by an organization's security policy. However, based on and results of data classification and on the security level attached to the data, there should be chosen the adequate cryptographic algorithms for encrypting the transmitted data. Essentially, there should be chosen the algorithms that have proven security and have not been yet broken yet. For achieving integrity, it is also recommendable to use HMAC SHA-1 algorithm because SHA-1 hash function requires more computational resources for being broken. HMAC-SHA1 is the more secure function, partly due to SHA-1's longer key length (SHA-1 uses a 160-bit key as opposed to the 128-bit key used by MD5). HMAC-MD5 is strong enough for a normal security environment, but HMAC-SHA1 is the better choice for a high-level security environment (see Chapter 4 for more details) (Zacker, 2006). When implementing IPsec, there could be enforced that both master key and encryption keys can be scheduled to change during data transmissions, and re-authentication can be required. Bragg (2003) pointed out that, generally, the more frequently the session is authenticated and the more frequently master and encryption keys are changed, the more secure the data will be. However, these options reduce the performance of the CPU. The options for using IPsec should be made based on sensitivity of data (we assume that the data is previously classified in the context of Jericho Project), and additional performance requirements should be met by additional hardware support. Northrup & Thomas (2004) stated that for IPsec implementations to be successful in large organizations IPSec policies must be deployed to all computers in organizations, and, also, the various methods used for deploying IPSec and the circumstances in which to use each method should be very well understood. Moreover, after IPsec has been implemented and deployed at a large scale for securing the communications, additionally, there should be implemented mechanisms for monitoring and troubleshooting IPsec. Monitoring IPSec is necessary for confirming that IPSec has been successfully deployed and is actively protecting communications. Although, IPsec is an open standard its use and configuration are still complex.

Comparison of IPsec and SSL/TLS security protocols

Both IPsec and SSL/TLS comprise security services that secure the communications over Internet. They ensure the authenticity, integrity, and confidentiality of the transmitted data. As we have already stated in this chapter, SSL/TLS and IPsec can be used to secure data transmissions in the context of Jericho Project. Generally, there are different situations when these protocols are considered appropriate for being used (Bragg, 2004):

When communications between specific computers can be defined, consider using an IPsec policy. When transmissions to and from specific ports should be absolutely blocked, IPsec blocking policies should be used. IPsec blocking policies are a way to create Internet Protocol filters. When transmissions to and from specific computers must be secured, IPsec should be used. SSL/TLS should be considered to secure communications for application supported by this protocol, between clients and servers, when sensitive information is transferred over Internet and needs to be secured. Essentially, Web server applications can be secured using SSL/TLS protocol. In the context of Jericho Project, SSL/TLS should be configured to require both server and client authentication.

SSL/TLS operates at the transport layer of the OSI model, and it is not transparent to applications. IPsec is a more universal security protocol because it operates at the network layer. Consequently, it is absolutely transparent for applications. IPsec is implemented by the operating system and is completely transparent to the applications that use IPsec. Resultantly, IPsec can be used to protect almost any type of network communication. IPsec can be used to encrypt all data without any need to redesign the application. SSL, however, must be designed into the individual application (Olifer & Olifer, 2007). Moreover, SSL/TLS does not secure all applications. Therefore, SSL/TLS cannot be used to encrypt all communications between two hosts. On the other hand, IPsec secures all the desired network communications independent of the interacting applications on the hosts. Additionally, IPsec provides connectionless security for communications. So, IPsec, unlike SSL/TLS, can secure connectionless communications such as UDP. SSL/TLS is less flexible than IPsec because it only supports authentication by means of public key certificates, while IPsec allows clients and servers to authenticate each other by using either public key certificates or a shared secret. Additionally, SSL/TLS allows one-way authentication, while IPSec requires both sides of a connection to authenticate. However, in the context of Jericho Project there should be specified and enforced mutual authentication, for both client and server, even when using SSL/TLS. Northrup & Thomas (2004), Alshamsi & Saito (2005) summarized the differences between IPsec and SSL/TLS as follows:

	IPsec	**SSL/TLS**
Authentication	Requires authentication for both the client and the server	Requires either the client or the server, or both, to be authenticated
Authentication Type	Authenticates by using either public-key certificates or a shared secret	Requires public-key certificate-based authentication

Applications supported	Can be used to authenticate and encrypt communications for any application	Can be used only to authenticate and encrypt communications for applications that specifically support SSL/TLS
Technology	Is a relatively a new technology that is not yet widely adopted	Is a mature technology that is widely adopted
Interoperability	Does not integrate well with other IPSec vendors, and in some cases modifications are required	SSL is trouble free and well integrated
MAC	Both IPSec and SSL/TLS require the implementation of HMAC-SHA-1 and HMAC-MD5-1 for authenticating the exchanged messages after the connection is established. HMAC- SHA-1-96→ 12 Byte HMAC-MD5-96 → 12 Byte	HMAC- SHA-1→ 20 Byte HMAC-MD5 → 16 Byte The strength of the Hash Algorithm is based on the length of the output.
Configuration	Hard	Easy
Pre-Shared Key	Yes	No
UDP support	Yes	No
Compression Support	Yes	OpenSSL only
Handshake Time	Slow	Fast

To conclude with, the following aspects should be considered when choosing between IPsec and SSL/TLS for securing the communications over Internet and achieving end-to-end encryption (Northrup & Thomas, 2004; Alshamsi & Saito, 2005; El Aoufi, 2006):

- IPsec can used to secure all IP traffic between computers; while SSL is specific to individual applications
- IPsec is transparent to applications, so it can be used with protocols that run on top of IP such as HTTP, FTP, and SMTP. But this can be regarded also as a concern, because when using IPSec approach, it provides too much isolation between the application and security services
- SSL/TLS is closely tied to the application
- IPSec can be used to ensure that only specific computers can connect to a server or can communicate with another host, in order to prevent attacks from other computers
- IPSec uses a shorter form of HMAC than SSL/TLS, thus SSL data integrity is more secure
- SSL is more compatible with firewalls than IPSec

- Unlike SSL/TLS, IPsec clients need special IPsec software to be installed
- Although sometimes compression is beneficial in data transmission, SSL/TLS does not support such a feature. IPsec supports compression
- In most cases IPSec does not interoperate well, so both sides of the connection are required to have the same vendor's IPsec deployment

The decision to use either IPSec or SSL for securing the communications depends on a number of factors. It depends on the users, the users' location, their reasons and needs for access, the device they are using, the services they request, the level of access they receive. Alshamsi & Saito (2005) concluded that choosing between IPSec and SSL depends on the security needs for a specific organization and, implicitly its users. When a user makes a request for a specific service and this is supported by SSL/TLS, it is better to select SSL/TLS because it eliminates the tasks of configuring, managing, and supporting IPSec client software installed on the users' computers, and it is easier and faster to deploy in comparison with IPsec. Furthermore, SSL/TLS is also included in standard Web browsers, such as Microsoft Internet Explorer and Mozilla FireFox. We can conclude that SSL/TLS technology has more advantages than IPSec and it started to be largely deployed and preferred for securing the communications. Nowadays SSL/TLS can support a wide range of Web-based applications apart from Web browsing and email, and it can be extended to support almost any IP-based application. Also, most applications for mobile workers are starting to be web-based for several reasons:

- Less training is required
- Application client software does not need to be installed on the users' PC
- The applications become available from any computer connected to the company's network

However, there are also IP-based applications that are not Web applications. But, the vendors of SSL/TLS technology have envisioned a few solutions to permit the use of legacy applications via SSL:

- Web interfaces to legacy applications
- Plug-ins to support specific applications

Using SSL/TLS protocol does no impose any specific software or hardware requirements, so it can be used from any computer connected to the Internet. However, when using SSL/TLS, because users can use any computer with a browser to access the Internet, certain policies should be specified and enforced by organizations for providing access to uncontrolled client computers1. El Aoufi (2006) presented the results published by Gartner with reference to a comparison between SSL/TLS and IPsec. Gartner considers that IPsec protocol will be replaced in the near future (till 2008) by SSL/TLS. Gartner predicts that IPsec will play a minor role in securing the communications, while, SSL/TLS will be the dominating technology for secure communications for most organizations transferring data over Internet. We conclude that for accomplishing the goals of secure communications in Jericho Project it is adequate to employ end-to-end security provided by SSL/TLS, and use IPSec only for applications that are not supported yet by SSL/TLS. Thus, IPSec will to be implemented for securing communications to unsupported applications, while SSL/TLS can be implemented as default for secure communications.

[1] In the context of Jericho Project, this issue is discussed as a separate research topic

XML Encryption

In the previous sub-chapters we have analysed SSL/TLS and IPsec protocols located at different layers in the OSI Model, as possible solutions in Jericho Project for providing secure communications within the scope of this project. As mentioned before, for data in transit, encryption is often the most appropriate means of ensuring confidentiality of communications. In this sub-chapter, we will shortly present the features of XML Encryption and the security services that could provide for ensuring secure communications within the scope of Jericho Project. Thorsteinson & Ganesh (2003) underlined the main differences between the security protocols that we have previously analyzed (SSL/TLS and IPsec) and that can be used to protect privacy and ensure integrity of data transmitted over Internet, and XML encryption. The authors pointed out two aspects of XML Encryption that contrast with the above presented cryptographic protocols:

- When using XML encryption, there can be selectively encrypted those XML elements that represent sensitive data, and other non-sensitive elements may be intentionally left unencrypted
- Moreover, XML encryption can be used for encrypting data that is either transmitted directly to another application or accessed by many applications via stored media, such as a disk file or database record

To the contrary, SSL/TLS and IPSec protocols encrypt the entire connection as a whole, allowing it to be used between two communicating entities. Anyway, XML Encryption does not replace these security protocols, but instead solves an entirely different type of security problem.

XML Encryption addresses two major issues:

- Encrypting only specific subsets of structured data
- Encrypting structured data storage that is accessible to multiple parties

Basically, encrypted data can be expressed in a structured manner using XML and portions of an XML document can be selectively encrypted. XML Encryption provides a standardized means for encrypting structured data and representing the result in a standard XML format. O'Neill et al. (2003) emphasized that selective encryption of an XML document is a new mechanisms introduced by XML Encryption. O'Neill et al. (2003) described the correspondence between Web Services and Services Oriented Architecture (SOA). The term "Services" in Web Services refers to a Service-Oriented Architecture (SOA). SOA is a recent development in distributed computing, in which applications call functionality (e.g. service, data) from other applications over a network (in the case of Jericho Project, the network is the Internet). In an SOA, functionality is "published" on a network where two important capabilities are provided— "discovery," the ability to find the functionality, and "binding," the ability to connect to the functionality. In the Web Services architecture, these activities correspond to three roles: Web Service provider, Web Service requester, and Web Service broker, which correspond to the "publish," "find," and "bind" aspects of a Service Oriented Architecture. There are Web Services technologies (e.g. WSDL, SOAP etc.) that enable SOA to run over Internet. Presenting these technologies is out of the scope of this book.

O'Neill et al. (2003) indicated the SOA publish/find/bind functionality in Web Services depends on XML. Rosenberg & Remy (2004) pointed out that XML is the foundation of the Web Services standards because XML is text-based and is designed to make business information transportable and self-describing. Further, we will describe the concepts that are used in for describing XML Encryption. World Wide Web Consortium (W3C) created a standard, business-centric data representation format, namely the Extensible Markup Language (XML). This is a meta language intended to supplement HTML's presentation features with the ability to describe the nature of the information being presented (Erl, 2004). XML is a derivative of the Standard General Markup Language (SGML). SGML is an international standard for defining electronic documents and represents a meta document definition language used for describing many document types with defining tags (Hartman et al., 2003). XML supplements the content of a document with meta information - self-descriptive labels for each piece of text. Thus, a Web document becomes a self-contained, mini-repository (Erl, 2004). XML Schema is a way of describing the rules for a particular XML document or instance. XML can be used as a manner to transport information as it passed between different computing systems. Object Access Protocol (SOAP) was defined as a way to transport XML from one computer to another. Observed in terms of a Service-Oriented Architecture, SOAP allows applications to bind to other applications for making use of their functionality. Essentially, SOAP provides a simple, consistent, yet extensible mechanism that allows one application to send an XML message to another application, so it can be regarded as a messaging protocol, as well as a means of using functionality that is published by a remote application (O'Neill et al., 2003; Rosenberg & Remy, 2004). Because XML is being used more and more to transmit data over Internet, it is vital to protect the transferred data in terms of privacy, integrity and data source authentication. Depending on the type of data and its security level, sometimes, the data can be transmitted in plaintext, but with the security services that ensure integrity of data and authentication of the data source. Thus, mechanisms, such as XML Encryption and XML digital signatures can be used for securing the transmitted data. Firstly, the same as in case of applying SSL/TLS or IPsec, the data has to be classified and should have attached a certain level of security, determining in this way how secure the data needs to be and what security mechanisms should be applied in order to protect it in traffic. XML Signature represents the underpinning technology for the standard called WS-Security and for Web services security in general. XML Signature is built on the mature digital signature technology. Digital signatures provide a mechanism for message integrity and non-repudiation. XML Signature enables the encoding of digital signatures into XML documents. XML Signature combines the utility and power of digital signature mechanism with the power and flexibility of XML. XML Encryption is built on the mature cryptographic mechanisms, specifically on symmetric-key cryptography. The core requirements for XML Encryption are that it must be able to encrypt an arbitrarily sized XML message, and it must do so efficiently. XML Encryption is employed basically for providing message confidentiality. In essence, XML Encryption represents a process for encrypting sensitive data and representing the result using the syntax of XML. O'Neill et al. (2003), Rosenberg & Remy (2004) mentioned that XML Encryption is appropriate to be used for Web Services, besides SSL/TLS or IPsec protocols, is because it allows the security principle of confidentiality to be satisfied across more than just the context of a single SOAP request. The security context of a SOAP message often extends beyond a single SOAP request. One obvious scenario is

if information in a SOAP message must be kept confidential while it is sent over a multi-hop SOAP transaction. In this scenario, if SSL alone is used, a gap exists at each SOAP endpoint, where the sensitive data would be temporarily in the clear. Moreover, if information in an XML message must be kept encrypted for confidentiality reasons, after the XML message has been processed by a Web Service, XML Encryption is also useful. This means that XML Encryption is what is called persistent encryption. This contrasts with session encryption (as in the case of SSL/TLS for instance). The encryption is not linked to the point-to-point SOAP exchange, so it does not end when the message reaches a SOAP endpoint. Thus, the key benefit XML Encryption is that it allows confidentiality to be satisfied across more than just the context of a single SOAP request. Moreover, it allows selective encryption and persistent encryption as well. XML Signature and XML Encryption apply standard cryptographic algorithms to data for achieving the desired security services for secure communications, and then store that encrypted and signed result in XML. Both mechanisms can be applied selectively only to portions of an XML document. Thorsteinson & Ganesh (2003) indicated that XML Encryption, the same as the other protocols used for securing the communications, makes use of a combination of symmetric and public-key cryptographic algorithms. The symmetric-key algorithms are used to encrypt the XML data elements, and the public-key algorithms are used to securely exchange the symmetric key used in the encryption process. The next table enumerates some of the algorithms and their identifiers used for XML security:

Algorithm	Type	Identifier
Triple DES	Block	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
AES-128	Block	http://www.w3.org/2001/04/xmlenc#aes128-cbc
AES-256	Block	http://www.w3.org/2001/04/xmlenc#aes256-cbc
AES-192	Block	http://www.w3.org/2001/04/xmlenc#aes192-cbc
RSA-v1.5	Key transport	http://www.w3.org/2001/04/xmlenc#rsa-1_5
RSA-OAEP	Key transport	http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
Diffie-Hellman	Key agreement	http://www.w3.org/2001/04/xmlenc#dh
Triple DES	Symmetric key wrap	http://www.w3.org/2001/04/xmlenc#kw-tripledes
AES-128	Symmetric key wrap	http://www.w3.org/2001/04/xmlenc#kw-aes128
AES-256	Symmetric key wrap	http://www.w3.org/2001/04/xmlenc#kw-aes256
AES-192	Symmetric key wrap	http://www.w3.org/2001/04/xmlenc#kw-aes192
SHA1	Message digest	http://www.w3.org/2000/09/xmldsig#sha1
SHA256	Message digest	http://www.w3.org/2000/09/xmldsig#sha256
SHA512	Message digest	http://www.w3.org/2000/09/xmldsig#sha512
RIPEMD-160	Message digest	http://www.w3.org/2001/04/xmlenc#ripemd160
Base64	Encoding	http://www.w3.org/2000/09/xmldsig#base64

In the view of Rosenberg & Remy (2004), XML Signature and XML Encryption are fundamental mechanisms for the next generation of emerging standards that use these two standards as building blocks. For instance, WS-Security, the emerging OASIS standard for Web services security, XML Key Management Specification (XKMS), and Security Assertion Markup Language (SAML), among many others, all rely on XML Signature and/or XML Encryption. W3C has developed specifications for encrypting and digitally signing XML. The XML-Signature Syntax and Processing specification[1] defines

[1] http://www.w3.org/TR/xmldsig-core/

processing rules and syntax to provide integrity, message authentication, and signer authentication services. The XML Encryption Syntax and Processing specification[1] defines a process for encrypting data and representing the result in an XML document. In the literature (O'Neill et al., 2003; Thorsteinson & Ganesh, 2003; Burnett, & Foster, 2004; Microsoft Corporation, 2005 etc.) described the process of achieving message layer security through using mechanisms such as XML Encryption and XML Signature. Moreover, they have presented the advantages of this method of achieving end-to-end security in comparison with SSL/TLS and IPsec protocols. Message layer security represents an approach used for achieving secure communications, in which all the information related to security is encapsulated in the message. In the context of Jericho Project XML encryption and XML Signature could be used when it is not necessary to encrypt and sign the entire data, but just the significant parts of it (in terms of security level and sensitivity) for achieving end-to-end security of the data in transit. For instance, XML Encryption allows encrypting a single, specific element of an XML document instead of encrypting the entire document or data, as would be the case in SSL/TLS and IPsec. Burnett & Foster (2004) pointed out that the resource benefit of XML Encryption that offer the option of encrypting only a small amount of data instead of the entire document becomes substantial. Moreover, the security provided by message layer security that uses XML Encryption regards the protection of the stored data as well, after it has been transmitted to the destination. While SSL/TLS and IPsec protect the data in only in traffic across the Internet, XML is a data-formatting specification that can be used to archive and store data as well, apart from protecting it in transit. Thus, we can conclude that message layer security is more flexible than the other security protocols we proposed in the context of Jericho Project for secure communications that incorporates all the security aspects of securing the data in transit in the message itself. This is in accordance with the Jericho Forum Commandments that specify that individual systems and data will need to be capable of protecting themselves in Jericho networks.

[1] http://www.w3.org/TR/xmlenc-core/

.

4. Cryptography

Cryptography is mainly used for sending information between different entities in such ways that others than the intended recipients cannot decipher it (Kaufman et al., 2002, p. 41). The scope of cryptography has diversified and became broader in the last years, mainly due to the advent of Internet. Currently, cryptography has a broader usage and applicability and includes cryptographic protocols, digital signatures along with the cryptographic algorithms and ciphers. In this chapter, firstly, we create a background for the research of cryptography, and then we present a more detailed approach for asymmetric key cryptography, symmetric key cryptography, elliptic curve cryptography, hash functions, MAC (message authentication code). Further more, we will emphasize the use of cryptography in the real world for offering certain security services and for designing security protocols that can be used within the scope of Jericho Project. We will begin our research in cryptography with the investigation of classical cipher and examples that incorporate fundamental security principles used in the modern ciphers as well. Then we will deal with more recent developed algorithms in cryptography, with attacks and methods to break cryptographic algorithms. The main purpose of this chapter is to offer a support and background knowledge for cryptographic primitives and protocols mentioned in the previous chapters for achieving the security requirements for secure communications in Jericho Project. Moreover, we aim at making recommendations regarding the cryptographic primitives that are the most adequate to use for end-to-end encryption in Jericho Project. Firstly, we will provide an overview of cryptography, and then symmetric and public – key algorithms will be presented. The cryptographic algorithms are divided into two groups: symmetric key algorithms and public-key algorithms. Symmetric key algorithms (secret-key algorithms) suppose that the same secret key is shared by the communicating parties for encryption and decryption. In public-key algorithms (asymmetric cryptographic algorithms), there are used two types of keys, a private key that is kept secret (e.g. for decryption, for signatures), and a public key that is made public. Further, hash functions are introduced, and then message authentication codes, followed by a presentation of elliptic curve cryptography. Finally, we will have a look at the current situation of cryptography in the real world and at the most appropriate algorithms that can be used for designing secure protocols in the context of Jericho Project.

Overview of cryptography

As Menezes et al. (1997), Mao (2003) stated, cryptography was used in the past (to be interpreted as the period before the 1960s), and is still used nowadays, as a tool for protecting the national secrets and strategies. Due to the advances in information technology, in the 1960s, there has started to appear the need for protecting the information in digital form also in the private sector. Although in the past cryptography was used exclusively for military and governments, nowadays, is largely deployed and underpins the security of the electronic world in civilian and corporate systems. The secure communications over Internet rely on the security features obtained with cryptography. Due to the explosion in communication technologies that erupted in the last decennia and to the wide use of the Internet, cryptography has become necessary (Atreya et al., 2002). The increasing computational power, parallel computing, new communication technologies increased the chances that different types of attacks can be performed on cryptographic algorithms. Therefore, the field of cryptography is also evolving. Mao (2003) underlined the need for the cryptography used in corporate

environments to adopt an open approach. The author stated that the cryptographic keys and the keying material should be kept secret, but the cryptographic algorithms and primitives should be made public for general review and, eventually, improvements. The same as required by Jericho Forum members, Mao (2003) acknowledged that in the areas of cryptographic algorithms, protocols and security systems, open research and standards, as well, are more than just a common means to acquire and advance knowledge. In Jericho Forum Commandment number 4 it is articulated that secure protocols demand open peer review to provide robust assessment and thus wide acceptance and use. According to Stamp (2006, p.2), cryptography has an important role to play in security protocols and represents a fundamental information security tool. Moreover, Vaudenay (2006) states that "cryptography is the science of information and communication security". Also, the same author defines cryptography as "the science of information protection against unauthorized parties by preventing unauthorized alteration of use." Stallings (2005) acknowledges that cryptography seems to be the most important aspect of communications security, and its importance increases in the field of computer security. In fact, cryptography comprises different mathematical techniques designed to protect communications. In this book we allocate also a considerate space for investigating, presenting and comparing different aspects of cryptography that are relevant and can further be used for Jericho Project for designing secure protocols in order to achieve secure communications over the Internet. "Cryptography itself has grown into an important branch of applied mathematics and theoretical computer science" (Dent & Mitchell, 2005, p.2). "Cryptography is the basic building block on which security principles such as authentication, integrity, non-repudiation and confidentiality are built" (modified De Laet & Schauwers, 2004; Ramachandran, 2002; Peter Gutmann[1]). According to the online publications of RSA Laboratories[2], cryptography can be defined briefly as the study of techniques and applications that depend on the existence of difficult problems. So, cryptography is fundamentally based on problems that are difficult to solve. A problem can be difficult to solve due to various reasons, for instance it requires secret knowledge to be able to find a solution or it is intrinsically difficult to complete etc. Difficult, in the context of cryptography, refers more to the computational requirements in finding a solution than the algorithmic and mathematical conception of the problem. These problems are called hard problems (in cryptography e.g. integer factoring, discrete logarithms, elliptic curve discrete logarithms). The role of a hard problem is to provide a security solution or service to users[3]. For instance, in the field of cryptography, encryption is used to provide confidentiality of data, and can also provide authentication and data integrity; digital signatures provide authentication, integrity and non-repudiation; hash functions provide integrity and can provide also authentication. Confidentiality means insuring that the information is protected against unauthorized users and is kept private. Cryptography can be also used to verify the integrity of a communication. Data integrity means ensuring that data has not been modified by unauthorized entities, and, thus, the message received by the recipient is the same as the message sent by the sender. Authentication provided through the means of cryptography can involve authentication of the entities involved in the communication process, and also data authentication

[1] Peter Gutmann' website http://www.cs.auckland.ac.nz/~pgut001/tutorial/index.html accessed April 2007

[2] http://www.rsa.com/rsalabs/node.asp?id=2157 accessed May 2007

[3] www.rsa.com/rsalabs/staff/bios/bkaliski/publications/other/kaliski-next-pkc-gt-1-2000.ppt accessed May 2007

origin (verify that the sender of the data is indeed the one who is supposed to be, and that is not being impersonated by an intruder). Non-repudiation ensures that the sender of any message cannot deny his/her actions. This can be achieved with digital signatures in conjunction with asymmetric key encryption (Solomon & Chappel, 2005). What is interesting about cryptography in the context of Jericho Project is the fact that basic cryptographic tools can be used to design and build cryptographic (security) protocols for secure communications over the Internet (e.g. transfer electronic money, authentication, end-to-end encryption etc.).` Before proceeding, we give some practical definitions for some basic cryptographic concepts.

- A message in the original form is named a plaintext or cleartext
- The mangled data is known as *ciphertext*
- A *cryptographic algorithm* converts a plaintext into a ciphertext
- The process of transforming a plaintext in ciphertext is named *encryption* in order to prevent any but the intended recipient from reading that data. The reverse of encryption is named *decryption*

According to Mao (2003), encryption is a practical way for achieving information secrecy. The author states that "Modern encryption techniques are mathematical transformations (algorithms) which treat messages as numbers or algebraic elements in a space and transform them between a region of "meaningful messages" and a region of "unintelligible messages".

- A *cipher* or *cryptosystem* is any method of encrypting text[1]. A *cipher* or *cryptosystem* is used to *encrypt* data

A thorough definition of a cryptosystem or a cryptographic system is provided by Menezes et al. (1996), Mao (2003), Oppliger (2005, p. 229).

Thus, a cryptographic system consists of:

- a plaintext message space ***M****:* a set of strings over some alphabet
- a ciphertext message space ***C****:* a set of possible ciphertext messages
- an encryption key space ***K****:* a set of possible encryption keys, and a decryption key space ***K'****:* a set of possible decryption keys

an efficient key generation algorithm $\boldsymbol{G}$: $\mathbf{N} \rightarrow \boldsymbol{K} \times \boldsymbol{K'}$
an efficient encryption algorithm $\boldsymbol{E}: \boldsymbol{M} \times \boldsymbol{K} \rightarrow \boldsymbol{C}$
an efficient decryption algorithm $\boldsymbol{D}: \boldsymbol{C} \times \boldsymbol{K'} \rightarrow \boldsymbol{M}$

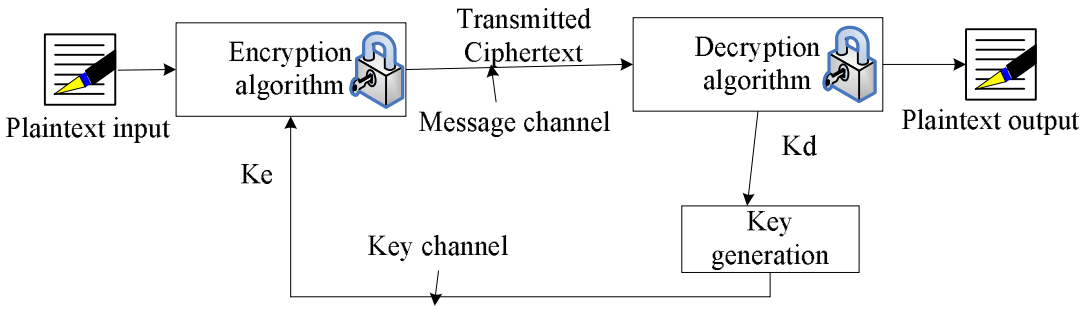
For instance, for integer 1, $\boldsymbol{G}(1)$ generates a key pair $(k_e, k_d) \in \boldsymbol{K} \times \boldsymbol{K'}$ of length $\boldsymbol{l}$.
For $k_e \in \boldsymbol{K}$ and $m \in \boldsymbol{M}$, there will result: $c = \boldsymbol{E}\, k_e(m)$ and this denotes the encryption transformation.

Further, $m = \boldsymbol{D}\, k_d(c)$ denotes the decryption transformation. It is though necessary that for all $m \in \boldsymbol{M}$ and all $k_e \in \boldsymbol{K}$, there exists $k_e \in \boldsymbol{K'}$.

$$\boldsymbol{D}\, k_d(\boldsymbol{E}\, k_e(m)) = m$$

[1] http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci213593,00.html accessed May 2007

Figure 12: Cryptographic system (adaptation after Mao, 2003)



As we mentioned earlier, there are two types of cryptosystems: secret-key cryptosystems (generally named symmetric-key cryptosystems in the literature) and public-key cryptosystems. The symmetric cryptosystems use the same key for encryption and decryption of the data (Ke=Kd). While, the public-key cryptosystems use a public key (Ke) for encryption and a private key (Kd) for decryption or for signatures. Oppliger (2005) provides the following definitions for secret-key and for public-key cryptosystems.

- A secret-key cryptosystem is a cryptographic system that uses secret parameters that are shared between the participating entities.
- A public-key cryptosystem is a cryptographic system that uses secret parameters that are not shared between the participating entities.

Stallings (2005) summarized the main characteristics of cryptographic systems according to three dimensions:

1) *The type of operations used for transforming plaintext to ciphertext*

The encryption algorithms are based on two general principles: *substitution,* in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and *transposition* or *permutation*, in which elements in the plaintext are rearranged. Most cryptographic systems involve multiple stages of substitutions and transpositions.

There are also variations of the operations mentioned above (substitution and transposition/permutation) that are discussed in the literature (Oppliger, 2005).

- *monoalphabetic substitution* cipher: each letter of the plaintext alphabet is replaced by another letter of the ciphertext alphabet; in this type of substitution, a plaintext letter is always replaced by the same ciphertext letter
- *homophonic substitution* cipher: plaintext letters can be replaced by more than one ciphertext letter. The letters that have a higher frequency in the plaintext are given more equivalents than the lower frequency letters
- *polyalphabetic substitution* ciphers: flatten the frequency distribution of ciphertext letters by using multiple ciphertext alphabets

2) *The number of keys used*

The system can be single-key or symmetric when the sender and the receiver use the same key. If the sender and receiver use different keys, the system is referred to as asymmetric or public-key encryption.

3) *The way in which the plaintext is processed*

According to this dimension, the ciphers are categorized in *block* and *stream ciphers*. Block and stream ciphers differ in how large a unit of the plaintext message is processed in each encryption or decryption operation. A unit may be either a bit or a block of bits (e.g. 64 or 128 bits). Block ciphers encrypt plaintext in blocks of different dimensions. Block ciphers take as input, messages that are precisely n-bits long and produce outputs of the same length (Dent & Mitchell, 2005; Seys, 2006). Common block sizes are 64 and 128 bits. While block ciphers operate on large blocks of data, stream ciphers typically operate on smaller units of plaintext, usually bits[1]. A stream cipher encrypts a plaintext one bit or one byte at a time (Stallings, 2005). Usually, a block cipher consists of a *round function* that is iterated several rounds. In each round, an appropriate transformation is applied using a subkey that is derived from the original secret key. With each performed round, the cryptanalysis of the cipher becomes more difficult, thus the security is improved. Nonetheless, with every performed round, the cipher becomes slower and more computations are required (Seys, 2006). The round function is typically a function of the output of the previous round and of a subkey which is a key dependent value calculated via a key scheduling algorithm. Dent & Mitchell (2005) remarked that the major problem with block ciphers is that the block length *n* is quite small and, typically, only short message can be encrypted with block ciphers. Also, if a block cipher used to encrypt the same plaintext message unit more times with the same encryption key, the same ciphertext block results every time. The block ciphers have different modes of operation[2] (Schneier, 1996; Menezes et al., 1997; Kaufman et al., 2002; Dent & Mitchell, 2005; Oppliger, 2005; etc). These modes of operation are used for plaintext messages exceeding one block in length. We give a short overview of these modes of operation in Section 4.2.4. According to the definition provided by Dent & Mitchell, a mode of operation for a block cipher is a method of using a block cipher iteratively to encipher long messages and at the same (generally) avoiding the problems associated with block ciphers (e.g. sending the same message twice, avoiding certain attacks based on statistical analysis). Menezes et al. (1997, p. 256) enumerated in their book some of the desirable characteristics for block ciphers. Some of these characteristics are:

- each bit of the ciphertext should depend on all bits of the key and all bits of the plaintext
- there should be no statistical relationship evident between plaintext and ciphertext
- altering any single plaintext or key bit should alter each ciphertext bit with probability 1/2
- altering a ciphertext bit should result in an unpredictable change to the recovered plaintext block

[1] http://www.rsa.com/rsalabs/node.asp?id=2174 accessed April 2007

[2] http://www.itl.nist.gov/fipspubs/fip81.htm DES Modes of Operation, FIPS PUB 81, National Bureau of Standards, U.S. Department of Commerce, 1981

In 1883, Kerckhoffs wrote a list of principles with reference to the cryptosystems (Menezes et al., 1996; Mao, 2003). From these principles, one became to be known as Kerchoffs' principle. This states that the knowledge of the algorithm and the key size, as well as availability of known plaintext are standard assumptions in modern cryptanalysis (Mao, 2003). An additional assumption in this case is that the attackers have access to all the communications over the ciphertext channel. So, the cryptographic strengths of cryptosystems should not be evaluated based on secrecy of the above mentioned elements. Also, Schneier (1996) stated that "a good cryptosystem is one in which all the security is inherent in knowledge of the key and none is inherent in knowledge of the algorithm."

Cryptographic Attacks

Cryptanalysis is the study of techniques that attempt to compromise, defeat or break cryptographic primitives[1]. Cryptography and cryptanalysis form together the field of cryptology. "Cryptographic attacks are designed to subvert the security of cryptographic algorithms, and they are used to attempt to decrypt data without prior access to a key" (Conrad, 2007). The cryptographic attacks belong to the field of cryptanalysis. The main purpose of attacks on cryptosystem is to recover the key in use, rather then simply to recover the plaintext of a single ciphertext. The types of attacks against encryption systems are discussed thoroughly in the literature (Schneier, 1996; Kaufman et al., 2002, p. 45; Mao, 2003; Rhee, 2003; Dent & Mitchell, 2005; Oppliger, 2005, p. 233-235; Stallings, 2005; Vaudney, 2006; etc.). The *cryptanalytic attacks* attempt to deduce partially or completely a plaintext or to deduce the key, based on the knowledge about the encryption algorithm and in some cases on some general characteristics of the plaintext. The attacker might even be in possession of some plaintext-ciphertext pairs. In *brute-force attacks*, the attacker tries every possible key on a part of ciphertext until finally a plaintext is obtained. Statistically, on average, half of all possible keys must be tried to achieve success. Further, we will introduce some of the widely known and studied cryptanalytic attacks.

[1] http://www.rsa.com/rsalabs/node.asp?id=2157 accessed April 2007

- *Ciphertext-only attack*

The cryptanalyst knows one or several ciphertext units and tries to deduce the corresponding plaintext message units and/or the key (or keys) that has (have) been used for encryption. If the attacker succeeds to recover the key (or keys), then he/she is able to decrypt any ciphertext encrypted with the key.

- *Known-plaintext attack*

The cryptanalyst has access to one or more pairs of <ciphertext, plaintext> and tries to deduce the key (or keys) used to encrypt the messages. The attacker tries to decrypt also other ciphertexts for which he/she does not have yet the corresponding plaintext.

- *Chosen-plaintext attack*

In this type of attack, the cryptanalyst has access to the encryption function or to the device that implements it. So, he/she can encrypt any plaintext at his/her choice. This is more powerful than a known-plaintext attack, because the cryptanalyst can choose specific plaintext messages to be encrypted and this could provide some additional information about the key. The attacker tries to deduce the key (or keys) used for encrypting the messages or to deduce an algorithm for decrypting new ciphers for which he/she does not possess the corresponding plaintext messages, but these have been encrypted with the same key (or keys). This type of attack is most common for public-key cryptography in the case that the attacker has access to the public key used for encryption. The chosen plaintext attack can be performed in two modes[1] (Schneier, 1996; Oppliger, 2005):

- *Batch chosen-plaintext attack*: The plaintext messages are chosen by the attacker beforehand encryption occurs.
- *Adaptive chosen-plaintext attack*: The attacker has not only the possibility of choosing plaintext messages for encryption, but also he/she can modify dynamically his/her choice of plaintext for further encryption based on the previous encryption results that were obtained.
- *Adaptive chosen-ciphertext attack*

Through this type of attack, the cryptanalyst has access to the decryption function (or the device that implements the function, respectively) and can decrypt any ciphertext unit of his/her choice. A device that provides decryptions of chosen ciphertexts units (either by accident or by design) is generically referred to as a "decryption oracle"[2]. The attacker tries to retrieve the key (or keys) that is (are) used for decryption or to determine the encryption scheme for being able to encrypt plaintext message units for which he/she does not have yet the corresponding ciphertext units. Also, this type of attack can be of two types:

- *Non-adaptive chosen-ciphertext attack*: This type of attack is named also indifferen chosen-ciphertext attack (or "lunchtime" attack). The ciphertext units are chosen before the decryption process begins. In the most successful attack scenario, this type of attack might successfully reveal the secret decryption key and thus completely break the scheme.

[1] http://en.wikipedia.org/wiki/Chosen_plaintext_attack accessed April 2007
[2] http://en.wikipedia.org/wiki/Chosen-ciphertext_attack accessed April 2007

- *Adaptive chosen-ciphertext attack*: This type of attack is know also as “”midnight” attack. The attacker can choose dynamically the ciphertext units for decryption, while the attack is performed. The results of the previous decrypted ciphertext units are used for selecting the subsequent ciphertext units, in order to gain information about encrypted messages and about the decryption key (or keys).

The cryptanalysts can use and exploit any combinations of these attacks in order to gain information about the encryption/decryption key (or keys). Nonetheless, Kerckoffs' principle applies without any doubt to the cryptosystems, namely, the encryption and decryption algorithms are assumed to be publicly known. If a cryptographic algorithm or its implementation is kept secret (known in the literature as security through obscurity) this does not make the respective algorithm unbreakable. In fact, the best algorithms that exist are the ones that have been made public (Schneier, 1996; Lail, 2002). These have been analyzed by the best cryptographers along the years and still remained unbreakable.

Public-key cryptography

In this sub-chapter we describe how public-key algorithms work. Furthermore, we perform an investigation regarding the security of the presented algorithms and the attacks to which they are exposed. In 1976, Whitfield Diffie and Martin Hellman published in an article (Diffie & Hellman, 1976) their work regarding public-key cryptosystem that changed the paradigm of cryptography forever (Schneier, 1996). In this article, the two authors described public-key cryptography and its applications. In public-key cryptography, the encryption, and, respectively, the decryption process are performed using two different keys, this is why it is also called asymmetric encryption. The encryption (public) keys can be made public. So, for instance, if the key is publicly available on Internet, anyone could theoretically encrypt a plaintext message with that key and send the ciphertext message to the owner of the private key. The entities participating in the communication process can exchange encrypted messages without any prior arrangement regarding the public key. The public-key algorithms have the following essential characteristic: It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key (Stallings, 2005). While, in symmetric-key cryptography the participants in any secure communication must have a prior ‘relationship’ because they must agree upon a common secret key that they will further use for encryption and decryption, with public-key cryptography it might be possible to securely exchange messages between entities with no prior ‘relationship’. Public-key cryptography can be used for more purposes: for exchanging between entities the secret keys used in symmetric key algorithms (key agreement & key management), for signing with the private key digital documents (authentication), and also for the same purposes of symmetric-key cryptography (encrypting data). The next Table illustrates the applications of some widely known public-key algorithms.

ALGORITHM	**Encryption/Decryption**	**Digital Signature**	**Key Exchange**
RSA	Yes	Yes	Yes
Diffie-Hellman	Yes[1]	No	Yes
Elliptic curve	Yes	Yes	Yes

Table: Applications for Public-key cryptosystems (modified after Stallings, 2005)

Stamp (2006) acknowledged that public-key cryptography has a critical role to play in modern information security. Usually, public-key cryptography and symmetric-key cryptography are used together in security protocols for achieving different security services. The resulting cryptosystems that combine secret- and public-key cryptography are often called hybrid. As we saw in Chapter 3 of this book, hybrid cryptosystems are used intensively within security protocols. Generally, as Schneider exposed in his book (1996), in most practical implementations public-key cryptography is used to secure and distribute session keys that are per session generated and used within symmetric-key algorithms for encrypting data in traffic. A public-key cryptosystem can be specified by a set of three algorithms: a probabilistic key generation algorithm, an encryption algorithm and a decryption algorithm.

Security of public-key cryptosystems

The security of a public-key cryptosystem is conditional on some assumptions that certain problems are intractable. In public-key cryptography a special attention is given to finding adequate protection against (adaptive) chosen-plaintext attacks, because the encryption key is public and these types of attacks are always possible and trivial to be performed. Thus, the design of cryptosystems that are resistant against these types of attacks receives a special importance in the research of public-key cryptosystems (Oppliger, 2005). In relation with the chosen-plaintext attacks, in public-key cryptosystems the concept of "semantic security against adaptive chosen-ciphertext attacks" has been introduced. For a cryptosystem to be semantically secure, it must be infeasible for an adversary to derive significant information about a plaintext message when given only its ciphertext and the corresponding public encryption key. Semantic security can be described also as indistinguishability of ciphertexts, meaning that the ciphertexts cannot be distinguished and consequently associated with plaintext messages. Another notion of security for public-key cryptosystems is non-malleability. An asymmetric encryption system is non-malleable if it is computationally infeasible to modify a ciphertext so that it has a predictable effect on the plaintext message. In the literature it has been shown that the notion of non-malleability is equivalent to the notion of semantic security against chosen-ciphertext attacks (Oppliger, 2005). In fact, the security of any cryptosystem depends on the length of the key and the computational work involved for breaking such a scheme (Stallings, 2005).

[1] According to Kaufman et al., 2002, p. 170

RSA

RSA public-key cryptosystem was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. RSA is a block cipher, where each block of plaintext has a binary value less than some number n. So, the block size must be less than or equal to $\log_2(n)$; in practice, the block size is i bits, where $2^i < n \leq 2^{i+1}$ (Stallings, 2005). RSA algorithm can be used for both public key encryption and digital signatures. The security of this algorithm is based on factoring large integers. Further, we describe RSA algorithm:

- Two large primes p and q are randomly chosen (Kaufman et al. (2002, p. 152) advised to choose them around 256 bits each); these numbers are kept private.
- $p \cdot q = n$ is computed; n is publicly known.
- n is the modulus for both the public and private keys
- The totient[1] $\Phi(n) = (p-1)(q-1)$ is computed
- An integer number e is chosen, which is relatively prime to $\Phi(n)$
- The public key for this algorithm is $\{e, n\}$
- For generating the private key, the number d should be found that is the multiplicative inverse of e mod $\Phi(n)$ ($d \cdot e = 1 \bmod \Phi(n)$) ; The relation $d \cdot e = k \cdot \Phi(n)$ should be satisfied for any integer k
- The private key for this algorithm is $\{d, n\}$
- Encryption of message m : $c = m^e \bmod n$
- Decryption of ciphertext c: $m = c^d \bmod n$

According to Stallings (2005), this algorithm has to meet the following requirements in order to be considered a satisfactory public-key cryptosystem:

- It is possible to find values of e, d, n such that $m^{ed} \bmod n = m$, for all $m < n$.
- It is relatively easy to calculate $m^e \bmod n$ and c^d, for all values $m < n$.
- It is infeasible to determine d given e and n.

In the literature (Kaufman et al., 2002) it is pointed out that RSA is not less secure if the number e, relatively prime to $(p-1)(q-1)$, is chosen to be the same number. If the number e is small or easy to compute, then the operations within RSA become more efficient. On the other hand, number d, which is the multiplicative inverse of e mod $\Phi(n)$, cannot be chosen to be small or a constant because of the possible attacks that might be conducted on such a private key. Kaufman et al. (2002) indicated two popular values for e that are 3 and 65537. These exponents work only if they are relatively prime to $\Phi(n)$.

Security of RSA

A brute force attack on RSA requires an exponential amount of overhead (Kaufman et al., 2002). The security of RSA is based on the difficulty of factoring big numbers. Stallings (2005) enumerated four types of attacks that can be conducted in order to break RSA:

- *Brute force*: This involves trying all possible private keys.

[1] The totient $\Phi(n)$ shows how many numbers are relatively prime to n. Relatively prime means that the numbers do not share any common factors except 1.

- *Mathematical attacks*: There are several approaches, all equivalent in effort to factoring the product of two primes.
- *Timing attacks*: These depend on the running time of the decryption algorithm.
- *Chosen ciphertext attacks*: This type of attack exploits properties of the RSA algorithm.

In addition to these mentioned attacks, there is always the risk of having side-channel attacks against a specific implementation of the algorithm. Stallings (2005) pointed out that the choice of a small constant value of *d* for efficient operation is not recommendable when using RSA algorithm. This is because, a small value of *d* is vulnerable to a brute-force attack and to other forms of cryptanalysis. So, there should be used a large key space when choosing the private key *d*. Thus, the larger the number of bits in *d*, the better. But, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower will run the system. Oppliger (2005) underlined as well the importance of the size of the public and private exponents, from a security point of view. As presented above as well, working with small private exponents is dangerous. In Table below there are enumerated the successful attempts of breaking RSA keys as responses to the challenges launched by RSA Laboratories. The level of effort is measured in MIPS-years: a million-instructions-per-second processor running for one year, which is about 3×10^{13} instructions executed. A 1 GHz Pentium is about a 250-MIPS machine.

Progress in Factorization (adapted after Stallings, 2005, RSA Laboratories Factoring Challenges[1])

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	Quadratic sieve
110	365	April 1992	75	Quadratic sieve
120	398	June 1993	830	Quadratic sieve
129	428	April 1994	5000	Quadratic sieve
130	431	April 1996	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003		Lattice sieve
174	576	December 2003		Lattice sieve
200	663	May 2005		Lattice sieve
	640	November 2005		

Stallings (2005) remarked that the threat to larger key sizes is twofold: the continuing increase in computing power, and the continuing refinement of factoring algorithms. The different algorithms used in the last years resulted in a tremendous speedup. The cryptanalysis of RSA regards more the task of factoring *n* into its two prime factors. Consequently, special attention should be engaged when choosing a key size for RSA

[1] http://www.rsa.com/rsalabs/node.asp?id=2092 accessed June 2007

algorithm. Stallings (2005) recommended that for the near future, a key size in the range of 1024 to 2048 bits seems reasonable. A key of 512 bits is no longer considered secure. For achieving more security there should be used keys of 2048 bits or even of 4096 bits. However, by using a large key size, the implementers and the users might get a false sense of security, which can raise other risks (e.g. side-channel attacks etc.). RSA Laboratories claimed that 1024-bit keys are likely to become broken some time between 2006 and 2010, and that 2048-bit keys are sufficient until 2030, although these results are under dispute in the cryptographic community. The author advised also to choose carefully the prime numbers *p* and *q*, in order to prevent the discovery of *p* and *q* by exhaustive methods. It is recommended that these primes are chosen from a sufficiently large set (*p* and *q* must be large numbers). Also Oppliger (2005) advised that *n* must be at least as large as to make it impossible to use an existing algorithm to factorize *n*.
Nowadays, there is a general consensus that at least 1024-bit moduli should be used. When specific parameters are recommended for a cryptographic algorithm, the value of the data must be taken into consideration. Thus, it can be recommended in certain cases to use 2,048-bit moduli for the asymmetric encryption of more sensitive or valuable data. If a 10240-bit moduli is chosen, the parameters *p* and *q* must be about 512 bits long each (Oppliger, 2005). Stallings (2005) enumerated a series of constraints that have been suggested previously by researchers. So, in order to avoid values of *n* that may be factored more easily, the algorithm's inventors suggested the following constraints on *p* and *q*:

- *p* and *q* should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both *p* and *q* should be on the order of magnitude of 10^{75} to 10^{100}
- Both (*p*-1) and (*q*-1) should contain a large prime factor.
- gcd(*p*-1, *q*-1) should be small.

Diffie-Hellman

Diffie-Hellman is the first public-key algorithm that was invented by Whitfield Diffie and Martin Hellman and made public in 1976. Diffie-Hellman public-key algorithm is not used for encryption or signatures, but mainly for exchange of secret keys. It allows entities to agree on a shared key. So, this algorithm is typically used for key exchange (named also key agreement, key negotiation, key distribution etc.).

.

Further, we shortly describe Diffie-Hellman key exchange protocol.

- Let *p* be a large prime and *g* be a generator of $\mathbf{Z}^*_p$
- Entities *A* and *B* (publicly) agree on *p* and *g*
- Entity *A* picks a large random number *x* and computes $X=g^x \bmod p$, where private exponent $x \in \{0, \ldots, p-1\}$
- *A* keeps number *x* a secret, but not the calculated value X (the public exponent). *A* sends X to *B*
- Entity *B* picks a large random number *y* and computes $Y=g^y \bmod p$, where private exponent $y \in \{0, \ldots, p-1\}$ (*x* and *y* should be 512-bit numbers chosen at random (Kaufman et al., 2002))
- *B* keeps number *y* a secret, but not the calculated value Y.

B sends Y to *A*

- *A* computes $K_{AB} = Y^x \bmod p = g^{yx} \bmod p$
- *B* computes $K_{BA} = X^y \bmod p = g^{xy} \bmod p$
- $K_{AB} = K_{BA} = K$ (by modular arithmetic)
- K is the secret key between entities *A* and *B*

It has been shown in the literature that, except entities *A* and *B*, the secret key K cannot be calculated by other parties that even possess the knowledge of $g^x \bmod p$ or $g^y \bmod p$. So, even if an eavesdropper would know g, p, X, Y, cannot calculate $g^{xy} \bmod p$. This problem is known as Diffie-Hellman Problem (DHP). Solving this problem is as difficult as solving the discrete logarithm problem (Kaufman et al., 2002; Oppliger, 2005).

Security of Diffie-Hellman

As we mentioned already, the security of a public-key cryptosystem is conditional on some assumptions that certain problems are intractable. The security of Diffie-Hellman is based on the difficulty of solving the discrete log problem. This problem is equivalently difficult with the problem on which RSA is based (difficulty of factoring). In its original description, Diffie-Hellman does not provide authentication of the parties, so it is exposed to the man-in-the-middle attack (Mao, 2003; Stallings, 2005). The attacker in the man-in-the-middle attack establishes two distinct Diffie-Hellman keys, one for communicating with *A* and the other for *B*. Further, it tries to masquerade as entity *A* for *B,* and as entity *B* for *A.* In fact, the attacker has total control of the communications between *A* and *B*. The solution for this type of attack is to introduce a method to authenticate the parties to each other. One of the solutions to prevent man-in-the-middle attack is, for each entity participating in the communication, to have already a permanent public number (e.g. X for *A*, Y for *B*) and the corresponding secret number that should be used for all the communications. In order for this technique to be usable, all the entities in the communicating set should previously agree on the common *p* and *g*. The generated public numbers for each entity are then published in a reliable manner (Kaufman et al., 2002). The generated public values, together with the global public values for *p* and *g*, are stored in some central directory. Whenever, user *A* wants to communicate with user *B*, *can* access user *B*'s public value, calculate a secret key, and use that to send an encrypted message to user *B*. If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication. In this case, only entities *A* and *B* can determine the key, confidentiality of the

communications is achieved. Entity *B* knows that only *A* could have sent the message using that key, and this provides authentication. This way of working for Diffie-Hellman protects it against active attacks. But this form of using authentication within Diffie-Hellman algorithm does not offer protection against replay attacks (Stallings, 2005). For combating the man-in-the-middle attacks on Diffie-Hellman, the entities can be authenticated to each other by means of public-key certificates or digital signatures. Another issue regarding the security of Diffie-Hellman algorithm refers to the publicly known values, *p* and *g*. In certain situations, when these numbers don't have some additional mathematical properties, the algorithm is less secure. For instance, it is desirable that also (*p-1*)/2 is also prime. A prime *p* that satisfies this additional constraint is called a safe prime or Sofie Germaine prime (Kaufman et al., 2002). Although it is computationally expensive to choose *p* and *g*, it is not advisable to use the same *p* and *g*.

Elliptic curve cryptography

Oppliger (2005) acknowledged that Elliptic curve cryptography (ECC) is a hot topic in contemporary cryptography. The algebraic structures employed by ECC are groups of points on elliptic curves defined over a finite field F_n. Thus, ECC makes use of elliptic curves in which the variables and coefficients are all restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications: prime curves over Z_p and binary curves over GF(2^m). An associative operation should be defined in order to make use of an elliptic curve. In ECC, this operation is called addition and signifies that two points on an elliptic curve are said to be added. However, this addition operation is explained geometrically in the literature dedicated to ECC. Stallings (2005) remarked that ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman. In the context of Jericho Project we aim to introduce the topic of ECC and to present the most interesting aspects regarding the security provided in comparison with the above presented public-key algorithms. Stallings (2005), Konheim (2007) defined a "hard problem" in order to form a cryptographic system using elliptic curves.

Consider the equation $Q = k \cdot P$ where $Q, P \in E_p(a, b)$ and $k < p$. It is relatively easy to calculate *Q* given *k* and *P*, but it is relatively hard to determine *k* given *Q* and *P*. This is named the discrete logarithm problem for elliptic curves.

For elliptic curve over Z_p, the following equation can be used for defining an elliptic curve:

$$y^2 \bmod p = (x^3+ax+b) \bmod p;\ 4a^3 + 27b^2 \neq 0 \pmod p$$

For elliptic curves over GF(2^m) there is used a cubic equation in which the variables and coefficients all take on values in GF(2^m), for some number m, and in which calculations are performed using the rules of arithmetic in GF(2^m).

$$y^2 + xy = (x^3+ax+b),\ a, b \in GF(2^m),\ b \neq 0.$$

Analog of Diffie–Hellman Key Exchange Using an Elliptic Curve

Stallings (2005), Konheim (2007) described how a key exchange occurs by using elliptic curves.

Key exchange using elliptic curves can be done in the following manner:

- A large integer *q* is chosen, which is either a prime number *p* or an integer of the form 2^m
- Then elliptic curve parameters *a* and *b* are chosen for the above mentioned equations used for elliptic curve cryptography. This defines the elliptic group of points $E_q(a, b)$
- Next, a base point $G = (x_1, y_1)$ is chosen in $E_p(a, b)$ whose order is a very large value *n*. The order *n* of a point G on an elliptic curve is the smallest positive integer *n* such that $n \cdot G = O$
- $E_q(a, b)$ and G are parameters of the cryptosystem known to all participants

A key exchange between entities *A* and *B* involves the following steps:

- *A* selects an integer n_A less than *n*. This is *A*'s private key. *A* then generates a public key $P_A = n_A \times G$; the public key is a point in $E_q(a, b)$.
- *B* selects a private key n_B and computes a public key $P_B = n_B \times G$
- *A* generates the secret key $K = n_A \times P_B$. *B* generates the secret key $K = n_B \times P_A$.

The two calculations performed by *A* and *B* for calculating the secret key, produce the same result :

$$n_A \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$$

To break this scheme, an attacker would need to be able to compute *k* given G and $k \cdot G$, which is assumed hard (Stallings, 2005). G, $n_A \times G$ and $n_B \times G$ are transmitted in the clear; a and b are secret. The secrecy of the Diffie–Hellman elliptic curve key exchange is the complexity of elliptic curve "integer" factorization (Konheim, 2007). Further, there can be defined an elliptic curve cryptosystem and an elliptic curve digital signature algorithm. However, due to the time and space allocated to this book we do not present these algorithms.

Security of ECC

Stallings (2005) specified that the security of ECC depends on how difficult it is to determine *k* given $k \cdot P$ and *P.* This is referred to as the elliptic curve logarithm problem. It is said that ECC provides the most security per bit when used for securing the communications. Oppliger (2005) mentioned that the elliptic curve cryptosystems are equally secure with smaller key sizes than their conventional counterparts, RSA and Diffie-Hellman. The advantage of ECC (pointed out by Certicom[1]) is that its inverse operation in the defined "hard problem" gets harder, faster, against increasing key length than do the inverse operations in DH and RSA. This signifies that as security requirements become more stringent, and as processing power gets cheaper and more available, ECC becomes the more practical system for use in the future for securing the communications. These properties of ECC are important for implementations in which key sizes and performance are important issues (e.g. smartcards). Further, at the end of this chapter we will provide a comparison of the presented public-key cryptography algorithms in terms of security offered and key length.

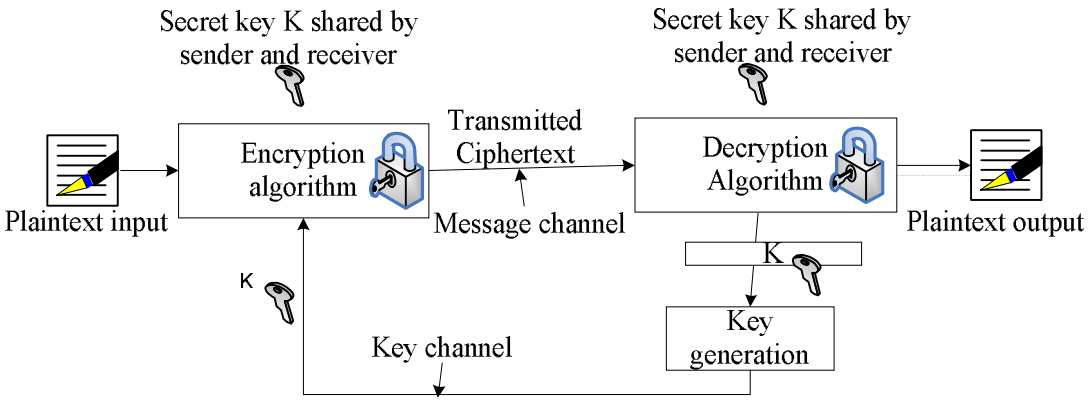
[1] http://www.certicom.com/

Symmetric-key cryptography

In this sub-chapter we describe how symmetric key algorithms work and the security services they offer for secure communications. As we already mentioned, in symmetric-key cryptography, the same key is used for encryption and decryption. The plaintext is encoded by mangling it with a secret key. The decryption process requires knowledge of the same key, and reverses the mangling. Using the same notation, symmetric-key cryptography can be represented as follows:

$$\boldsymbol{D}\, k_d(\boldsymbol{E}\, k_e(m)) = m, \text{ with the condition that } k_d = k_e$$

Figure 13: Simplified model for symmetric-key cryptography



The decryption algorithm is the reverse of the encryption algorithm. For decryption, the encryption algorithm is run in reverse on the ciphertext using the secret key. In order for symmetric-key cryptography to be used securely there are two basic requirements that have to be taken into consideration (Stallings, 2005). Firstly, a strong encryption algorithm should be used, in the sense that even if an attacker has knowledge of the algorithm and has intercepted a ciphertext, would still be unable to decrypt it or retrieve the key with the information available at hand. In a more restrictive form, this requirement establishes that the attacker should be unable to discover the secret key even if it has access to different ciphertexts and to the corresponding plaintexts. Secondly, the secret key should be distributed in a secure manner to the entities involved in the communication process. As we mentioned in Section 4.1, there are certain operations that are performed on the plaintext messages for transforming it in the ciphertext. Substitutions and permutations are used in the design of symmetric cryptosystems in order to obtain confusion and diffusion. In the design of symmetric encryption systems, permutations and substitutions are usually used and combined (sometimes in multiple rounds) to provide confusion and diffusion (Schneier, 1996; Menezes et al., 1997, p. 20; Oppliger, 2005, p. 238; Stallings, 2005). The terms confusion and diffusion were introduced by Claude Shanon in 1949 in his article "Communication Theory of Secrecy Systems". These two techniques are intended to offer protection against cryptanalysis attackes based on statistical analysis. In diffusion, the statistical structure of the plaintext is distributed into long-range statistics of the ciphertext. The influence of a single plaintext bit is spread among several ciphertext

bits. The patterns, structures or redundancies in the plaintext are dissipated in the ciphertext. As Stallings (2005) observed, the purpose of diffusion is to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key.

The mechanism of confusion is intended to make the relationship between the encryption key and the statistics of the ciphertext as complex as possible in order to stop the attempts of discovering the key. The symmetric-key cryptosystems are mainly designed and used for minimizing the computations required to encrypt and decrypt the message units. Possible drawbacks of these cryptosystem can arise from the flaws in securely distributing the secret key between the communicating parties. This poses great risks in the case that attackers can obtain access to secret key or keys (Jaworski & Perrone, 2000). Further on, we will describe some of the most representative symmetric-key algorithms.

Data Encryption Standard (DES)

Data Encryption Standard (DES) is a symmetric cipher defined in Federal Information Processing (FIPS) Standard Number 46[1] in 1977 as the federal government approved encryption algorithm for sensitive but non-classified information. In the literature (Schneier, 1996; Menezes et al., 1997; Kaufman, 2002; Mao, 2003; Rhee, 2003; Oppliger, 2005; Stallings, 2005; etc.) there are allocated considerable spaces for the thorough description of the DES algorithm. DES was developed by IBM in the 1970s and was based upon the previous research done by IBM for Lucifer cipher. DES algorithm was adopted as adopted in 1977 as the Data Encryption Standard by the National Bureau of Standards (NBS) as FIPS PUB 46. Nowadays, the FIPS PUBS are developed and maintained by the National Institute of Standards and Technology (NIST). NIST recommended the use of DES for applications other than the protection of classified information. The standard (DES) was reaffirmed in 1983, 1988, 1993, and 1999, and it was officially withdrawn in July 2004[2]. The last reaffirmation in 1999[3] of the Data Encryption Standard contains the specifications of DES and of *Triple Data encryption Algorithm* (TDEA) that can be used for protecting highly sensitive data (Oppliger, 2005; p. 239). Typically, the triple DEA algorithm is referred as *triple DES* (3DES). Firstly, we will have a look at DES, and then, an overview of 3DES will be provided later in this chapter. The design of DES is based on two general concepts: product ciphers and Feistel ciphers (Menezes et al., 1997, p. 250). Further, these ciphers are shortly explained. A product cipher refers to the design of a complex encryption function by composing several simple operations (transpositions, translations (e.g. XOR) and linear transformations, arithmetic operations, simple substitutions etc.) that used together offer complementary, but individually insufficient protection. The definition of product cipher, provided by Menezes et al. (1997, p. 251), states that it combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components.

Feistel Ciphers

[1] http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf accessed May 2007
[2] http://csrc.nist.gov/Federal-register/July26-2004-FR-DES-Notice.pdf accessed May 2007
[3] http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf accessed May 2007

Feistel (1973) proposed the use of a cipher that alternates substitutions and permutations on block plaintexts (Menezes et al., 1997; Oppliger, 2005; Stallings, 2005; etc.). For Feistel cipher, the alphabet is $\Sigma = \mathbf{Z}_2 = \{0, 1\}$, and the block length is $2t$ (for a reasonably sized $t \in \mathbf{N}^+$). The Feistel cipher runs in $r \in \mathbf{N}^+$ rounds. For every $k \in K$, there are generate r round sub-keys $k_1, \ldots, k_r$ that are used on a per-round basis. In general, the sub-keys k_i are different from K and from each other (Stallings, 2005). Firstly, the encryption function, E_k, divides the plaintext message block m into two halves of t bits each. We consider, L_0 be the left half, and R_0 be the right half, so a message m has for instance the structure $m = (L_0,R_0)$. The two halves of the plaintext message unit pass through r rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs L_{i+1} and R_{i-1}, derived from the previous round, as well as a sub-key k_i.

A sequence of pairs (L_i,R_i), for $i = 1, \ldots, r$, is then recursively computed as follows:

$$(L_i,R_i) = (R_{i-1}, L_{i-1} \oplus f_{ki} (R_{i-1}))$$

This means that $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f_{ki} (R_{i-1})$.

For $i = 1$, then L_1 and R_1 are computed as follows:

$$L_1 = R_0$$

$$R_1 = L_0 \oplus f_{k1}(R_0)$$

For the last round, r, L_r and R_r are computed as follows:

$$L_r = R_{r-1}$$

$$R_r = L_{-1} \oplus f_{kr} (R_{r-1})$$

The cipher block is represented by the pair (R_r,L_r), rather than (L_r,R_r),. Thus, the encryption of plaintext message m using key k can formally be expressed as follows:

$E_k(m) = E_k(L_0,R_0) = (R_r, L_r)$

In Feistel cipher, for recursive computation of L_i and R_i, the following formula can be used:

$$(L_{i-1},R_{i-1}) = (Ri \oplus f_{ki}(Li), Li).$$

The decryption of Feistel cipher uses the same encryption algorithm by applying the round keys in reverse order, $k_r, \ldots, k_1$. For the decryption process, the input is formed by the ciphertext and the round keys in reverse order.

In Figure 14 (page 109) there is illustrated the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm (after Stallings, 2005). The following notation is used: LE_i and RE_i represent the block units processed through the encryption algorithm, and LD_i and RD_i for the lock units processed through the decryption algorithm. Regarding the relation between the encryption and decryption block units at each round in Feistel cipher, Stallings (2005) made the following observation: considering the *output* of the i^{th} encryption round being $LE_i \,||\, RE_i$ (with the signification LE_i concatenated with RE_i), then the corresponding *input* to the 16^{th} decryption round is $RE_i \,||\, LE_i$, or equivalently $RD_{16-i} \,||\, LD_{16-i}$.

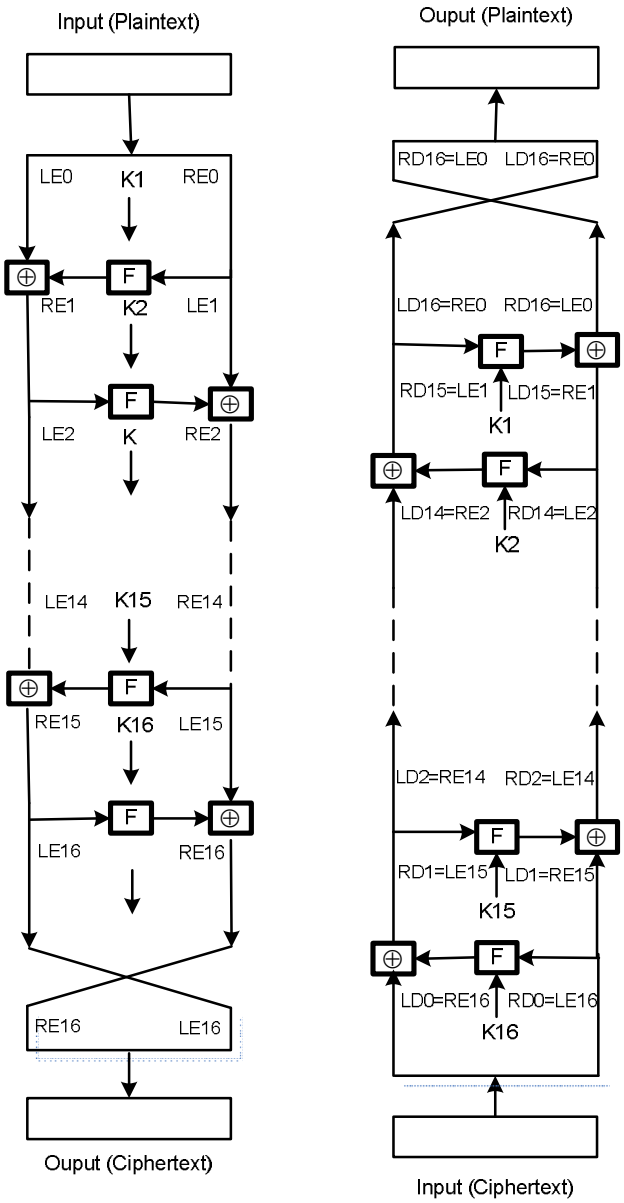


Figure 14: Feistel Encryption and Decryption (adapted after Stallings, 2005)

DES Description

DES is a symmetric block cipher that operates on 64-bit blocks and uses a 56-bit key. DES encrypts data in blocks of 64 bits. Basically, DES is a Feistel cipher with *t*=32 (so, the block length is 64 bit) and *r*=16 (for rounds) (Oppliger, 2005, p. 241).The DES encryption and decryption algorithms operate in 16 rounds. The input to the algorithm is a 64-bit block of plaintext and the output from the algorithm is a 64-bit block of ciphertext after 16 rounds of identical operations. The key length is 56 bits by removing the 8 parity bits. Every eighth bit from the initial 64-bit key is used for parity checking and is ignored. The basic building block of DES is a suitable combination of permutation and substitution on the plaintext block. DES applies the same combination of techniques on the plaintext block 16 times. Typically, for an encryption scheme there are two inputs to the encryption function: the plaintext to be encrypted and the key. For DES as well, the encryption function will have as input 64-bit plaintext message units to be encrypted and the 56-bit key. In DES, the processing of the plaintext proceeds in three phases (Stallings, 2005). Firstly, the 64-bit plaintext block X is transposed under

the initial permutation IP, resulting X_0 = IP(X) = (L_0,R_0). Then, the following phase consists of 16 rounds of the same function that involved both permutation and substitution operations. The output of the last round (the 16th) consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the pre-output. Finally, the pre-output is passed through a permutation (IP^{-1}) that is the inverse of the initial permutation function, to generate the 64-bit ciphertext block Y.

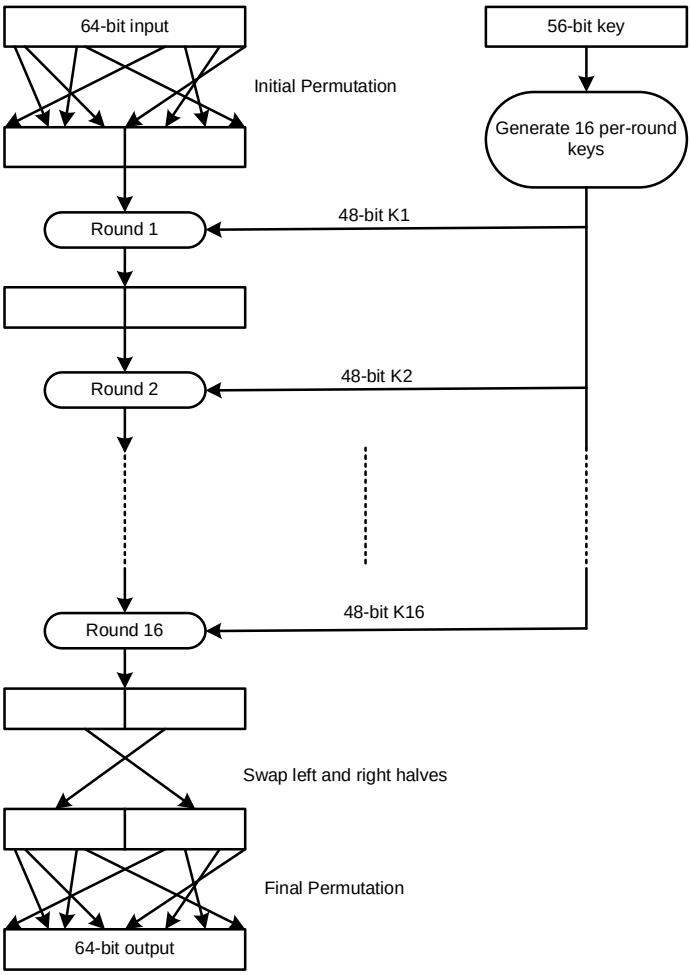


Figure 15: Overview of the basic structure of DES (adapted after Kaufman et al., 2002, p. 65)

Further, we provide a short explanation of the DES encryption scheme. Firstly, the 64-bit input will pass through an initial permutation (IP(*m*)) from which results a 64-bit shuffled input. The 56-bit key is used to generate 16 keys of 48-bit per-round. This is done by selecting different 48-bit subsets from the initial 56-bit key for each round key. The input of each round consists of 64-bit output of the previous round and the 48-bit per-round key; then, an output a 64-bit is produced.

DES encryption consists of 16 rounds. After IP(*m*) is performed, a 16 round Feistel cipher is applied to IP(*m*). The 64-bit input plaintext is divided into 32-bit halves L_0 and R_0. The rounds are functionally equivalent, 32-bit inputs L_{i-1} and R_{i-1} are used from the previous round and the 32-bit outputs L_i and R_i for $1 \leq i \leq 16$ are generated as follows:
$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, where $f(R_{i-1}; K_i) = P(S(E(R_{i-1}) \oplus K_i))$
- E represents a fixed expansion permutation
- P is another fixed permutation on 32 bits

After the last round, the left and right halves are exchanged and, finally, the resulting message is bit-permuted by the inverse of IP. After 16 rounds are performed, the left and the right part of the final 64-bit output are swapped, and then another permutation is performed on this 64-bit message unit by the inverse of IP. Decryption involves the same key and algorithm, but with subkeys applied to the internal rounds in the reverse order. The decryption process for DES algorithm works by running DES algorithm backwards (Kaufman et al., 2002, p. 65). It has been analyzed in the literature the initial and the final permutation on the data included in DES do not increase the security of the algorithm and have no apparent cryptographic significance, but one reason might be to make DES less efficient to be implemented in software (Kaufman et al., 2002, p.66; Mao, 2003). For instance, the Initial Permutation (IP) is a fixed function (e.g. is not parameterized by the input key) and is also publicly known.

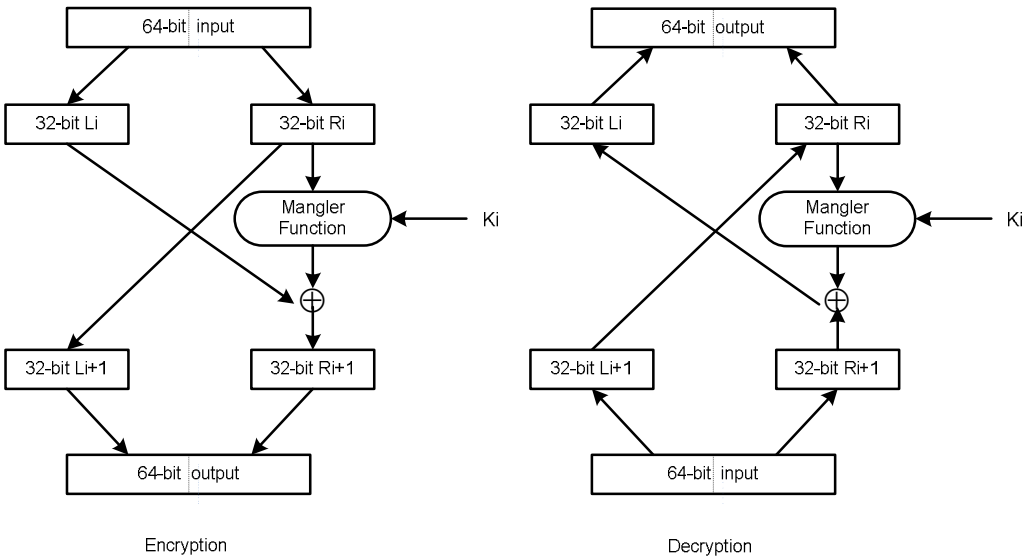


Figure 16: How the encryption and decryption algorithms work in a DES round

In encryption process in one DES round, the following actions occur:

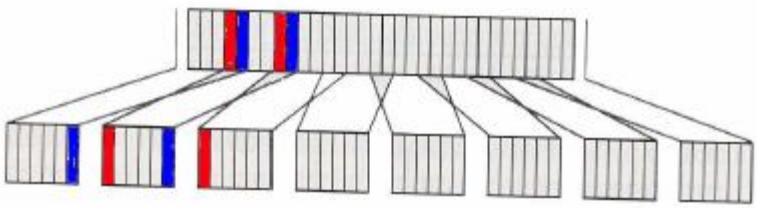
- the 64-bit input is divided into two 32-bit halves L_i and R_i for $1 \leq i \leq 16$
- outputs L_{i+1} and R_{i+1} are generated
- then L_{i+1} and R_{i+1} are concatenated and results the 64-bit output of the round
- in a DES round, L_{i+1} is R_i
- R_{i+1} is obtained as follows: L_i and K_i constitute the input to a mangler function; the function generates a 32-bit output; then, the operation "⊕" is applied to this 32-bit output together with L_i, and results R_{i+1}

In decryption process in one DES round, the following actions occur:

- the 64-bit input is formed by L_{i+1} concatenated with R_{i+1}, and these two halves are known; we want to determine L_i and R_i
- R_i is L_{+1}; K_i is also known
- we know already that $R_{i+1} = L_i \oplus \text{mangler}(R_i, K_i)$; we want to determine L_i
- mangler(R_i, K_i) is computed, then the output is "⊕" with R_{i+1} , and it results L_i
- The mangler function is not reversible, although DES algorithm is reversible
- R_{i+1} is obtained as follows: L_i and K_n constitute the input to a mangler function; the function generates a 32-bit output; then, the operation "⊕" is applied to this 32-bit output together with L_i, and results R_{i+1}

The mangler function takes as input the 32 bits of data (R_i) and the 48-bit key (K_i,) and generates a 32-bit output. The operation exclusive-or (⊕, module 2 or XOR) is applied between the output of the mangler function and L_i, and it results R_{i+1}. Firstly, mangler function expands R_i to 48-bit value, by divining it into 4-bit chunks that are then expanded to 6-bit chunks by taking the adjacent bits and concatenating them to the chunks (Kaufman et al., 2002). For this, an expansion function is being used $E : \{0, 1\}^{32} \rightarrow \{0, 1\}^{48}$.

Figure 17: Expansion of R_i to 48-bit value



Further, the 48-bit K_i is split also in chunks of 6 bits. Then the corresponding chunks of K_i are ⊕ with the corresponding 6-bit chunks of the expanded R_i; then the output of 6 bits value constitutes the input for S-box (a function $S_i : \{0, 1\}^6 \longrightarrow \{0, 1\}^4$). The S-box maps several input values to the same output value. The 4-bit output of each of the eight S-boxes is concatenated into a 32-bit output that is subject to a permutation before being ⊕'d with L_i. The result is R_{i+1}.

Security of DES

Talbot & Welsh (2006, p. 117) stated that even now the best known practical attack on DES is by brute force, that is searching through all 2^{56} possible keys. In 1998, the Electronic Frontier Foundation[1] built a machine that succeeded in decrypting a DES ciphertext message after approximately three days. Oppliger (2005, p. 250) affirmed that the most serious vulnerabilities and security problems of DES are caused due to the relatively small key length and the corresponding feasibility of an exhaustive key search. The author suggests some solutions for protecting a block cipher with a small key length, such as: the keys can be frequently changed, or a complex key set up procedure can be used. The simplest suggested method for protecting a block cipher against exhaustive key search attacks is to use sufficiently long keys. For solving the problem of small key length for DES, there are three possibilities:

- The DES may be modified in a way that compensates for its relatively small key length
- The DES may be iterated multiple times (this resulted in TDEA or 3DES)
- An alternative symmetric encryption system with a larger key length may be used (e.g. AES)

DES was still reaffirmed as a federal standard in 1999, despite the fact that a brute force attack on DES was already known to be feasible. However, it was then recommended that a variant known as Triple DES be used instead. Because DES has been subject to public scrutiny since 1977, it has been shown that there are sixteen DES keys that have certain properties and should be avoided to be used (Kaufman, 2002, p. 74; Oppliger, 2005, p. 247; Menezes et al., 1997, p. 257). There are 4 weak keys and 12 semi-weak keys, and these keys are less secure than other keys.. Still, the probability to generate one of these keys is $16/2^{56}$, which, in the opinion of cryptography literature authors (Schneier, 1996; Kaufman, 2002, p. 74; Oppliger, 2005, p. 248), is negligible. A DES key k is weak if $DES_k(DES_k(m)) = m$ for all $m \in \boldsymbol{M} = \{0, 1\}^{64}$, meaning that the DES encryption with k is inverse to itself (e.g. if m is encrypted twice with a weak key, then the result is again m). The DES keys k_1 and k_2 are semi-weak if $DES_{k1}(DES_{k2}(m)) = m$ for all $m \in \boldsymbol{M} = \{0, 1\}^{64}$, meaning that the DES encryptions with k_1 and k_2 are inverse to each other.

3DES

An alternative to solve the problem of small key length for DES is to use multiple encryption with DES and multiple keys. Multiple iterations have to be done with different keys in order to improve security of the algorithm. It has been shown in the literature (Rhee, 2003; Oppliger, 2005; Stallings, 2005) that given any encryption keys K_1 and K_2, it would not be possible to find a third key K_3 such that:

$$E(K_2,E(K_1,\mathrm{m}))=E(K_3,\mathrm{m})$$

When DES is iterated two times using two encryption keys it becomes vulnerable to the man-in-the-middle attack. This supposes that an adversary has in possession some (plaintext, ciphertext) pairs (m_i, c_i), c_i is derived from a double encryption of m_i with K_1 and K_2, and he/she wants to find K_1 and K_2. The attack is thoroughly described by Oppliger (2005), Stallings (2005). This is one of the reasons why DES should be iterated 3 times, resulting in the use of 3DES. As we already mentioned, 3DES or Triple

[1] http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/ accessed May 2007

Data Encryption Algorithm (TDEA) is specified by FIPS PUB 46-3[1]. 3DES can be used alternatively with two keys ($K_1 = K_3$, K_2) or with three keys (K_1, K_2, K_3). The last version is preferred, as it results in a great increase in cryptographic strength (Rhee, 2003). Another option, mentioned by Oppliger (2005, p. 255), is to consider all three keys equal ($K_1 = K_3 = K_2$), this representing a single-key DES implementation. The 3DES encryption process works in the following way:

$$c = E(K_3, (D (K_2, E(K_1,m))),$$

which is an EDE ("encrypt-decrypt-encrypt") process.

Stallings (2005) mentioned that 3DES with three keys would be a preferred alternative encryption algorithm by the researchers due to the cryptographic strength given by the 168-bit key length. The authored stated also that 3DES. Moreover, Stallings (2005) stated that 3DES is advantageous from two points of view. Firstly, the fact that is has 168-bit key length overcomes the vulnerability to brute-force attack of DES. Secondly, the encryption algorithm in 3DES is the same as in DES. The underlying algorithm of DES has been subjected to more scrutiny than any other encryption algorithm over a longer period of time, and no effective cryptanalytic attack based on the algorithm rather than brute force has been found. Consequently, there is a high level of confidence that 3DES is very resistant to cryptanalysis.

International Data Encryption Algorithm (IDEA)

International Data Encryption Algorithm (IDEA) originally being called Improved Proposed Encryption Standard (IPES), was developed in the early nineties by Xuejia Lai and James Massey at Swiss Federal Institute of Technology in Zurich (ETHZ). IDEA encrypts 64-bit plaintext unit blocks into 64-bit ciphertext unit blocks using a 128-bit key. The same algorithm is used for encryption and decryption. IDEA operates also in rounds as DES. But each primitive operation in IDEA maps two 16-bit quantities into a 16-bit quantity, whereas each DES S-box maps a 6-bit quantity into a 4-bit quantity. This cryptosystem contains three primitive operations and they are all easy to be computed in software for creating a mapping and are also reversible (for decryption). The operations used within IDEA are: XOR (bitwise exclusive or "$\oplus$"), addition modulo 2^{16}, multiply modulo $2^{16}+1$. The result has to be always 16 bits and this is not always the case when adding or multiplying two 16-bit quantities. The algorithm expands the 128-bit key into 52 16-bit keys. The key expansion is executed differently for encryption than for decryption, but the encryption ad decryption operations are the same in this cryptosystem (Kaufman et al., 2002). The decryption sub-keys are either the additive or multiplicative inverses of the encryption sub-keys. IDEA has eight rounds that can be treated also as sixteen rounds (odd rounds and even rounds). In the literature (Schneier, 1996; Menezes et al., 1997; Kaufman et al., 2002), IDEA algorithm is described thoroughly. For IDEA some classes of weak keys have been found (Daemen et al., 1994), but they are so rare so there is no need to avoid them explicitly. It can be attempted to break IDEA by exhaustive search on 128-bit key space, but this requires unbelievable computing resources (Kaufman et al., 2002). In 2003, Demirci et al. presented their findings about a "new man-in-the-middle attack" on the reduced-round versions of IDEA block cipher. Also, they mentioned in their paper other kind of attacks that have been applied by other authors on reduced-round versions of IDEA. In addition, Demirci et al. (2003) made a review of the previous attacks on IDEA and their

[1] http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf accessed May 2007

complexity, the number of rounds, the number of chosen plaintexts needed in order to conduct the respective attacks, the necessary memory.

Advanced Encryption Standard (AES)

In January 1997, National Institute of Standards and Technology (NIST) launched another competition for selecting a new encryption standard to be used for protecting sensitive, non-classified U.S. government information. After close examination of different submissions, NIST chose an algorithm named Rijndael named so after the cryptographers that designed it (Joan Daemen & Vincent Rijmen).

The NIST criteria[1] for evaluating AES algorithm for becoming a new standard are summarized in the next table:

Security	**Actual security:** compared to other submitted algorithms (at the same key and block size). **Randomness:** the extent to which the algorithm output is indistinguishable from a random permutation on the input block. **Soundness:** of the mathematical basis for the algorithm's security. **Other security factors:** refer to any attacks that demonstrate that the actual security of the algorithm is less than the strength claimed by the submitter.
Cost	**Licensing requirements:** the algorithm(s) specified in the AES shall be available on a worldwide, non-exclusive, royalty-free basis. **Computational efficiency:** The evaluation of computational efficiency will be applicable to both hardware and software implementations. Computational efficiency essentially refers to the speed of the algorithm. **Memory requirements:** The memory required to implement a candidate algorithm for both hardware and software implementations of the algorithm.
Algorithm and Implementation Characteristics	**Flexibility:** Candidate algorithms with greater flexibility will meet the needs of more users than less flexible ones, and therefore, inter alia, are preferable. However, some extremes of functionality are of little practical application (e.g., extremely short key lengths); for those cases, preference will not be given. Some examples of flexibility may include (but are not limited to) the following: **a.** The algorithm can accommodate additional key- and block-sizes (e.g., 64-bit block sizes, key sizes other than those specified in the Minimum Acceptability Requirements section, [e.g., keys between 128 and 256 that are multiples of 32 bits, etc.]) **b.** The algorithm can be implemented securely and efficiently in a wide variety of platforms and applications (e.g., 8-bit processors, ATM networks, voice & satellite communications, HDTV, B-ISDN, etc.). **c.** The algorithm can be implemented as a stream cipher, message authentication code (MAC) generator, pseudorandom number generator, hashing algorithm, etc. **Hardware and software suitability:** A candidate algorithm shall not be restrictive in the sense that it can only be implemented in hardware. If one can also implement the algorithm efficiently in firmware, then this will be an advantage in the area of flexibility. **Simplicity:** A candidate algorithm shall be judged according to relative simplicity of design.

In 2000, there were defined more specifically other evaluation criteria for Rijndael cipher. AES is intended to replace 3DES more because of efficiency reasons. But still, 3DES remains an approved algorithm for use in the future. The new cryptographic algorithm, AES, became a Federal Information Processing Standard[2] in November 2001, and became effective in May 2002, under the name Advanced Encryption

[1] http://csrc.nist.gov/CryptoToolkit/aes/round1/aes_9809.htm

[2] FIPS 197 contains the actual specification of AES http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Standard. Rijndael is a symmetric block cipher with variable block and key length. The block and key length can be chosen independently from 128, 160, 192, 224, and 256 bits. For AES the block-length was fixed to 128-bit and three different key sizes (128, 192 and 256-bits) were specified. In the case of a brute force attack on AES cipher, an exhaustive search on 2^{128} (= 3.4 x 10^{38}) possible 128-bit keys, 2^{192} (= 6.2 x 10^{57}) possible 192-bit keys, and 2^{256} (= 1.1 x 10^{77}) possible 256-bit keys should be performed[1].
Stallings (2005) declared that Rijndael was designed to have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

Similarly to DES, AES is an iterated block cipher with a block length of 128 bits and a variable key length of 128, 192, or 256 bits. Therefore, AES-128, AES-192 and AES-256 resulted as three different versions of AES. The number of rounds for each version of AES depends on the key length (e.g. 10, 12 or 14 rounds).

Although initially it has been publicly informed by NIST[2] that AES can be used for protecting unclassified sensitive information, in 2003 it has been announced by The Committee on National Security Systems[3] U.S. that National Security Agency (NSA) agreed that AES algorithm (for all the key length e.g. 128, 192, 256 bits) can be used to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 bits key lengths. Either, NSA suspected fundamental weakness in keys that have a shorter length, or they might give preference to a safety margin for top secret documents.

This can be summarized as showed:

	N_b	N_k	N_r
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

- N_b represents the block length (the number of 32-bit words in an encryption block)
- N_k refers to the key size in 32-bit words
- N_r represents the number of rounds

The number of rounds, N_r, is a function of the other two parameters (N_b and N_k). It needs to be larger for longer keys, so it would become as difficult to break the encryption algorithm as it would be to perform a brute-force attack to recover the respective key. This allows also sufficient mixing in the encryption process, such that each bit of the plaintext block has a complex effect on the resulting ciphertext block (Kaufman et al., 2002).

[1] http://csrc.nist.gov/CryptoToolkit/aes/aesfact.html accessed May 2007
[2] FIPS 197 contains the actual specification of AES http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf accessed May 2007
[3] http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf accessed May 2007

.

Rijndael specifies the following formula for the number of rounds:

$$N_r = 6 + \max(N_b, N_k)$$

The official versions of the AES all work with a block size of $N_b \cdot 32 = 4 \cdot 32 = 128$ bits. Despite this, some authors (El Aoufi, 2006, p. 50-51; Stamp, 2006, p. 46) presented also the cases of AES algorithm with different block lengths (e.g. 128, 192, 256 bits).

This is summarized in Table at page 115 that complements the results presented in Table below.

Block size (words/bytes/bits) / **Key size** (words/bytes/bits)	4/16/128	6/24/192	8/32/256
4/16/128	10 rounds	12 rounds	14 rounds
6/24/192	12 rounds	12 rounds	14 rounds
8/32/256	14 rounds	14 rounds	14 rounds

The formula $N_r = 6 + \max(N_b, N_k)$ that represents the way of calculating the number of rounds holds for larger block lengths for AES cipher. Internally, AES operates on a two-dimensional array of bytes and this represents its state. The state has 4 rows and N_b 4-octet (32 bits) columns. The input to the encryption and decryption algorithms is a 128-bit block. The input block is represented as a square matrix of bytes that is further transmitted in the state array. The state array is modified at each round of encryption or decryption. After the final round, state is copied column by column in the output matrix. Rounds 1 to N_r -1 comprise an identical sequence of operations (see further the description), while in round N_r one operation is omitted. The key that is a 4 N_k-octet block is also depicted as a square matrix of bytes[1]. The key is further divided into N_k 4-octet columns. Then additional columns are created until $(N_r+1)\cdot N_b$ number of columns are reached, this representing the exact amount of expanded key required. Rijndael is based on the following primitive operations:

1. *Substitute bytes:* Uses an S-box to perform a byte-by-byte substitution of the block
The bytes of the state are substituted according to a given substitution table (this transformation is called SubBytes() in the AES specification).

[1] Note that the ordering of bytes within a matrix is by column.

2. *ShiftRows:* This represents a simple permutation
The rows of the state are shifted left by different offsets (this transformation is called ShiftRows() in the AES specification).

3. *MixColumns:* A substitution that makes use of arithmetic over GF(2^8)
The data within each column of the State are mixed (this transformation is called MixColumns() in the AES specification).

4. *AddRoundKey:* A simple bitwise XOR of the current block with a portion of the expanded key
A round key is added to the state (this transformation is called AddRoundKey() in the AES specification).

It is worthy to note that the SubBytes() and ShiftRows() transformations commute. This implies that if a SubBytes() transformation is immediately followed by a ShiftRows() transformation, this is equivalent to a ShiftRows() transformation immediately followed by a SubBytes() transformation (Oppliger, 2005). Each round, except the final round, comprises the four different operations mentioned above that are considered internal functions to be described and performed in a moment (Mao, 2003). In the final round, the MixColumns() operation is omitted. The round transformations are invertible for the purpose of decryption. Mao (2003) analyzed in detail these primitive operations and he concluded that they are quite simple, so their implementation can be done with extremely good efficiency.

Basic Structure of AES encryption and decryption (adapted after Kaufman et al., 2002; Stallings, 2005; El Aoufi, 2006)

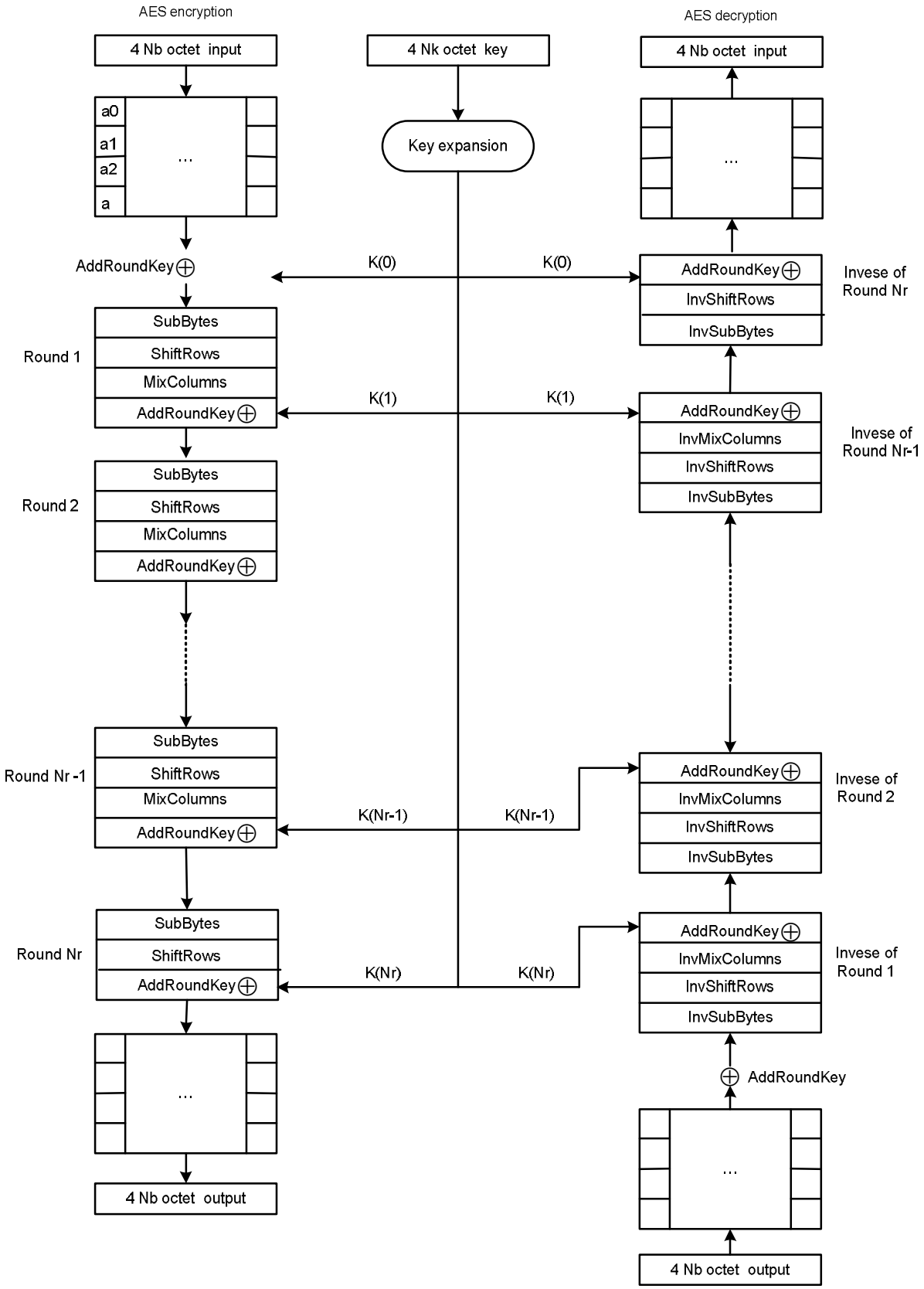


Figure 17: The basic structure of AES endcryption algoritm

As we can observe from Figure 17, AES decryption can be implemented by applying the inverse of the primitive operations in opposite sequence from that in the encryption algorithm. The individual primitive operations used in the AES decryption algorithm are

named InvShiftRows(), InvSubBytes(), InvMixColumns(), and AddRoundKey(). The AddRoundKey() transformation is its own inverse, as it only involves a bitwise addition modulo 2. As the SubBytes() and ShiftRows() operations commute, this holds also for their inverse InvSubBytes() and InvShiftRows() operations.

Security Analysis of AES

AES is still believed to be very secure[1] due to the fact that it has been designed to resist against classical approximation attacks, such as linear cryptanalysis, differential cryptanalysis. Despite this, nowadays there have been reported some successful attacks against Rijndael/AES algorithm (Schneier, 2005; El Aoufi, 2006). Some research groups of Institute for Applies Information Processing and Communication have reported and summarized on their website[2] the latest security aspects regarding AES algorithm. The attacks against Rijndael/AES algorithm, and generally against block ciphers, are usually performed on some slightly adjusted versions of the cryptosystems that have fewer rounds than the official versions. For instance, attacks that succeeded to break the AES cipher were executed on versions with 6, 7 and 9 rounds.

The best known short-cut attacks on each of the three official AES versions (Oswald et al., 2002, p.1; Al Aoufi, 2006, p. 67)

Attack	**Year**	**Paper**	**AES-128** 10 Rounds	**AES-192** 12 Rounds	**AES-256** 14 Rounds
Related Key	2005	Biham et al. (2005)		9 rounds	
Truncated Differential	2003	Jakimoski & Desmedt (2004)		6 rounds	
Impossible Differential Related-Key	2003	Jakimoski & Desmedt (2004)		8 rounds	
Impossible Differential	2001	Cheon et al. (2001)	6 rounds		
Square Attack	2000	Lucks (2000)		7 Rounds	7 Rounds
Square Attack	2000	Ferguson et al. (2000)	7 Rounds	7 Rounds	9 Rounds
Collision Attack	2000	Gilbert & Minier (2000)	7 Rounds	7 Rounds	7 Rounds

As far as we know up to now, these are the attacks reported in the literature on AES algorithm. The reports about these attacks raise worries among cryptographers regarding the security of AES. The margin between the number of rounds specified by the cipher and the rounds of the best known attacks is becoming smaller and smaller for achieving and maintaining the desired level of security. The risks reside in the possibility of finding ways to improve these attacks so that AES algorithm can be broken.

Some researchers[3] revealed a new type of attacks on block ciphers, namely on Rijndael/AES. These attacks that are named algebraic attacks are based on the algebraic and mathematical structure of the cryptosystems. Currently, these new

[1] http://www.cryptosystem.net/aes/ accessed May 2007
[2] http://www.iaik.tugraz.at/research/krypto/AES/index.php accessed May 2007
[3] http://www.cryptosystem.net/aes/ accessed May 2007

algebraic attacks are not practically applicable for Rijndael algorithm due to the complicated calculations that should be done. But research is further conducted in this direction. Other types of attacks against Rijndael/AES algorithms are those referred as side- channel attacks. Side channel attacks do not perform attacks on the underlying cipher, but rather they attack based on information gained from the physical implementation of the cipher on systems which inadvertently leak data. Examples of side-channel attacks are the timing attacks, power analysis, fault analysis. These attacks make assumptions about the implementations of the ciphers and use additional information gained from attacking those implementations. Timing attacks assume that an attacker knows the relative time a particular encryption operation takes. In the literature[1] (Bernstein, 2005; Osvik et al., 2005; etc.) there are presented different side-channel attacks on AES algorithm and the research conducted in the field of side-channel analysis.

Camellia

Camellia supports 128-bit block size and 128, 192, and 256-bit key lengths. Camellia was developed jointly by Nippon Telegraph and Telephone Corporation (NTT) and Mitsubishi Electric Corporation in 2000. It was designed to withstand all known cryptanalytic attacks, and it has been scrutinized by worldwide cryptographic experts. Camellia is internationally recognized as the unique 128-bit block cipher that possesses the security level and processing capability equivalent to AES. Moreover, Camellia was selected as the EU recommended cipher and E-government recommended cipher in 2003 and was also adopted as the ISO/IEC international standard cipher. Due to its security features, Camellia is proposed for implementation also in the design of security protocols such as SSL/TLS, IPsec and XML Encryption.

RC4

RC4 is one of the most popular symmetric stream ciphers. Stream ciphers use internal state, thus the i^{th} ciphertext unit depends on the i^{th} plaintext unit, the secret key, and some state. Stream ciphers are of two types: synchronous (or additive) stream ciphers and non-synchronous (or self-synchronizing) stream ciphers. Typically, a stream cipher generates a one-time pad and applies it to a stream of plaintext by using the operation $\oplus$ (XOR or addition modulo 2). A one-time pad is a long random (or pseudo-random) string of characters or numbers that is generated and further is (one-time) used for encrypting a message with $\oplus$ operation (Kaufman et al., 2002, p.92). Messages encrypted with keys based on randomness have the advantage that there is theoretically no way to "break the cipher" by analyzing a succession of messages. Each encryption process is unique and is not chained to the next encryptions. RC4[2] is a variable-key-size additive stream cipher and it was developed in 1987 by Ron Rivest for RSA Data Security, Inc. Although the algorithm was not indented to be publicly disclosed by being a trade secret of RSA Data Security, Inc., in 1994 it was posted anonymously on a mailing list, and in this way it became rapidly widely available on Internet. Actually, RC4 represents a simple and fast generator of sequences of pseudorandom bytes (e.g. a key stream) that are generated independently from the plaintext messages or ciphertext, and further these sequences are added modulo 2 ($\oplus$) to the plaintext messages byte sequence (Kaufman et al., 2002; Oppliger, 2005). The

[1] http://www.iaik.tugraz.at/research/krypto/AES/index.php#sca accessed May 2007

[2] The acronym RC stands for "Ron's Code"

cipher generates variable-length key that can range from 1 to 256 bytes (2048 bits). Oppliger(2005) describes in detail how the key stream is generated. Following, this is the description of the algorithm. RC4 uses an array *S* of 256 bytes of state information (called S-box). The elements of *S* are labelled S[0], . . . , S[255] and they are initialized in three steps:

1. All elements of *S* are initialized with their index:

S[0] = 0
S[1] = 1
. . .
S[255] = 255

2. Another array S_2 of 256 bytes is allocated and filled with the key, repeating bytes as necessary.

3. The S-box is then initialized as suggested in the S-Box initialization algorithm (see below). The S-Box initialization algorithm only operates on *S* .

(*S*)
for *i* = 0 to 255 do
j ← (*j* + *S*[*i*] + *S*2[*i*]) mod 256
S[*i*] ↔ *S*[*j*] // the S-box entries *S*[*i*] and *S*[*j*] are swapped
(*S*)

In addition, after *S* is initialized (according to algorithm presented above), *i* and *j* are set to zero (all entries of S_2 are also set to zero). Furthermore, RC4 key generation algorithm (see below) is used in order to generate a potentially infinite sequence of key bytes. The algorithm takes *S* as input parameter and outputs a key byte *k*.

(*S, i, j*)
i ← (*i* + 1) mod 256
j ← (*j* + *S*[*i*]) mod 256
S[*i*] ← *S*[*j*]
t ← (*S*[*i*] + *S*[*j*]) mod 256
k ← *S*[*t*]
(*k*)

If a plaintext message (ciphertext) of *l* bytes must be encrypted (decrypted), then the algorithm must be iterated *l* times, and each key byte k_i (*i* = 1, . . . , *l*) must be added modulo 2 to the corresponding plaintext message (ciphertext) byte.

Security of RC4

Stallings (2005) pointed out that this cryptosystem is subject to a series of attacks that make it vulnerable in front of different attacks. Thus, RC4 is not recommended for use in new applications. Another aspect that can make RC4 is to use the same keystream to encrypt two different documents. If the same keystream is used to encrypt different plaintexts, the encryption can be broken by XORing the two ciphertext streams together. The keystream drops out, and the result represents a plaintext XORed with another plaintext. The plaintexts can be recovered by using letter frequency analysis

and other basic techniques[1]. However, generally, the primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers.

Block Cipher Modes of Operation

In the sub-chapters above, we have presented different symmetric cryptosystems (DES, IDEA, AES) that encrypt blocks of messages. For effectively using these ciphers, there have been proposed different modes of operation[2] that specify different modalities of encrypting multiple block messages with a block cipher. By using different modes of operation longer messages can be encrypted as well. A cryptographic mode of operation usually combines the cipher, a sort of feedback mechanism, and some simple operations (Schneier, 1996). The security of the encryption process resides in the cryptosystem that is being used and not in the mode of operation.

1. Electronic Code Book (ECB)

This is the simplest mode of operation in which a plaintext message is split into 64-bit blocks (the last block is padded out, if necessary, to be 64 bits). If the block at the end is shorter, it has to be padded with "some regular pattern" (e.g. zeros, ones, alternating ones and zeros) in order to make it a complete block of 64 bits. Each block is encrypted at a time with the same secret key. The message blocks can be encrypted in any order because they are independent. Each encrypted block is decrypted separately. With this mode of operation, theoretically, it can be created a code book because it creates a fixed mapping between plaintexts and corresponding ciphertexts. Each plaintext block has a corresponding ciphertext block for a certain encryption key. This makes this operation mode vulnerable to attacks of adversaries. The ECB mode of operation has the following properties:

- If a message contains identical plaintext blocks, these will result in identical ciphertext when encrypted with the same key. So, an attacker can gain information from similar ciphertext blocks

- The blocks can be encrypted and decrypted independently, so if the ciphertext blocks are decrypted and reorder accordingly, they result in the corresponding plaintext. So, an attacker can modify and re-arrange the blocks differently in his/her advantage if this is the case.

Schneier (1996) presented ECB mode's weaknesses in examples, and showed that an adversary can easily remove, repeat, or interchange blocks in the communication process if he/she is able to intercept the transmitted messages. This can be prevented by using message authentication codes (MACs), or changing frequently the encryption keys or by using another operation mode for the cipher block (chaining).
ECB should not be used in communications between different entities for encrypting messages due to its mentioned disadvantages.

[1] http://www.schneier.com/blog/archives/2005/01/microsoft_rc4_f.html accessed June 2007
[2] DES Modes of Operation FIPS PUB 81 http://www.itl.nist.gov/fipspubs/fip81.htm accessed May 2007

2. *Cipher Block Chaining (CBC)*
   The cipher block chaining mode is removing some of the disadvantages of ECB. For instance, if the same plaintext block occurs several times in the encryption process, the resulting ciphertext blocks are different in the chaining mode. In the cipher block chaining, the input for the encryption algorithm is result of the XOR of the current plaintext block and the preceding ciphertext block. The same secret key is used for each block. The encryption of a plaintext block depends not only on the current block and the key, but also on the previous plaintext blocks and the initialization vector (IV) used for the first plaintext block. For the first plaintext block, a random number (IV) is chosen and is ⊕'d with the plaintext, and the result is encrypted under the secret key. With CBC mode, identical plaintext messages are mapped to different ciphertext blocks due to the fact that the input for the encryption function is different for every plaintext message.
   Using the notation from the beginning of the chapter, the encryption formula for a plaintext m_i in CBC is:
   $c_i = E(m_i \oplus c_{i-1}, K)$, for i = 1, 2, . . .
   and the decryption:
   $m_i = D(c_i, K) \oplus c_{i-1}$, for i = 1, 2, . . .
   For the first block the encryption is:
   $c_0 = E(m_0 \oplus \text{IV}, K)$,
   and the decryption:
   $m_0 = D(c_0, K) \oplus \text{IV}$

   In the literature (Schneier, 1996; Kaufman, 2002), it is mentioned that the IV should be random in order to prevent the generation of similar ciphertext blocks for similar plaintexts blocks. This hinders the attackers to build code books or to replay block messages in the communication process. Also, there is no need to keep the initialization vector secret and this should be regarded as a dummy initial ciphertext block in the chaining process. Stallings (2005) suggested that IV should be protected against unauthorized changes. With CBC mode, attackers cannot determine if repeated values are transmitted, but they can still modify the ciphertext. As error propagation is concerned, with CBC, a single bit error in a plaintext block will affect the corresponding ciphertext block and all subsequent ciphertext blocks. But, the decryption process reverses the effect, and the resulting plaintext message has the same error in the end. If there is a single bit error in the ciphertext, this affects one block and one bit of the corresponding plaintext. The plaintext block that contains the error is completely garbled[1], the plaintext block after contains only one bit error in the same bit position as the error in the cipher, and further the propagation of the error stops. The system recovers and continues to work correctly for all subsequent blocks.

3. *Cipher Feedback Mode (CFB)*
   The cipher feedback mode turns the block cipher into a stream cipher. A stream

[1] An error in transmission, reception, encryption, or decryption that changes the text of a message or any portion thereof in such a manner that it is incorrect or cannot be decrypted.

cipher eliminates the need to pad a plaintext message to be a multiple of a certain number of bits. Schneier (1996) mentioned that some applications need to send in real time messages in a communication process, for instance bit by bit, or byte by byte. This can be done with OFB and CFB modes that can encrypt data in units smaller that the used cipher block size. CFB can be used for encrypting 64 bits plaintext at once or any *k*-bit CFB, where *k* is less than or equal to the block size. Dent & Mitchell (2005, p.81) advised that it is better to choose the number of plaintext bits being encrypted equal with the number of bits of the block cipher. This is recommended by the second edition of ISO/IEC 10116[1], and any other choice would appear to reduce the overall level of security of the scheme. Schneier (1996) pointed out that the initialization vector (IV) for CFB mode has to be unique for each transmitted message, otherwise, the attackers can easily recover the plaintext message. In CFB mode, an error in the plaintext affects all subsequent ciphertext and reverses itself at decryption. Typically, for *k*-bit CFB, a single ciphertext error affects the decryption of the current and subsequent *n*/*k*-1 blocks, where *n* is the block size for the cipher. So, if *k* is a larger number, fewer errors are propagated. The transmitted ciphertext error will be discarded sooner from the register. A disadvantage for CFB mode regards the performance of the encryption/decryption process. For instance, if DES is used in CFB mode with plaintext blocks of 8 bits, then the performance is 8 time slower than DES on 64-bit plaintext blocks. An initial vector (IV) is used with the block cipher for generating a sequence of pseudorandom bits. Then, *k* bits (the left most significant bits) of the output of the encryption function are selected and XORed ($\oplus$) with *k* bits from the plaintext (typically k=8 for one byte or k=1 for one bit). In this way, the first unit of ciphertext c_1 is generated. At the beginning of the encryption process, the plaintext message is split initially into blocks of *k* bits each. Further more, the contents of the shift register are shifted left by *k* bits and c_1 is transferred in the rightmost (least significant) *k* bits of the shift register. This process continues until all plaintext blocks have been finally encrypted. For the decryption process, the same encryption function is used to generate the *k* bits output that is then XORed with the corresponding ciphertext block for generating the plaintext block (Stallings, 2005).

[1] http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=38761&scopelist=PROGRAMME accessed May 2007

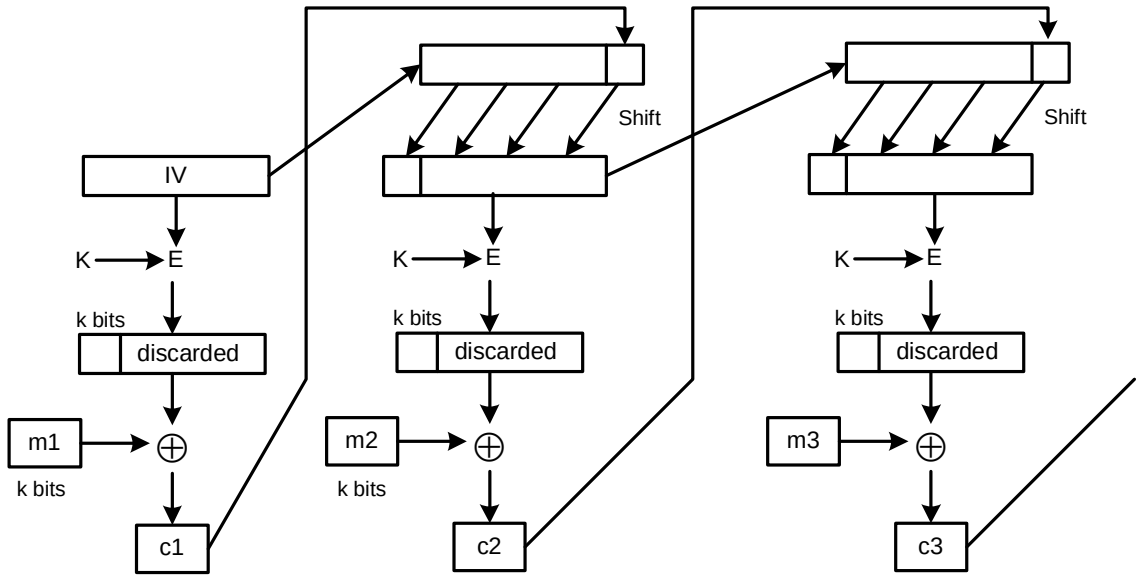


Figure 18: The encryption process in CFB mode (after Kaufman et al., 2002; Stallings, 2005)

4. *Output Feedback Mode (OFB)*
   Output-feedback mode is a method of transforming a block cipher in a stream cipher. It is somehow similar to CFB, and allows encryption of various block sizes. But, in the case of output feedback mode, the *k* bits (the left most significant bits) of the output of the encryption function are selected and serve as the feedback. They are added at the right to the shift register. These feedback blocks form a one-time pad (string of bits) that can be used as a key stream that is XORed with the plaintext blocks. The one-time pad can be generated in advance before the message to be encrypted is known.

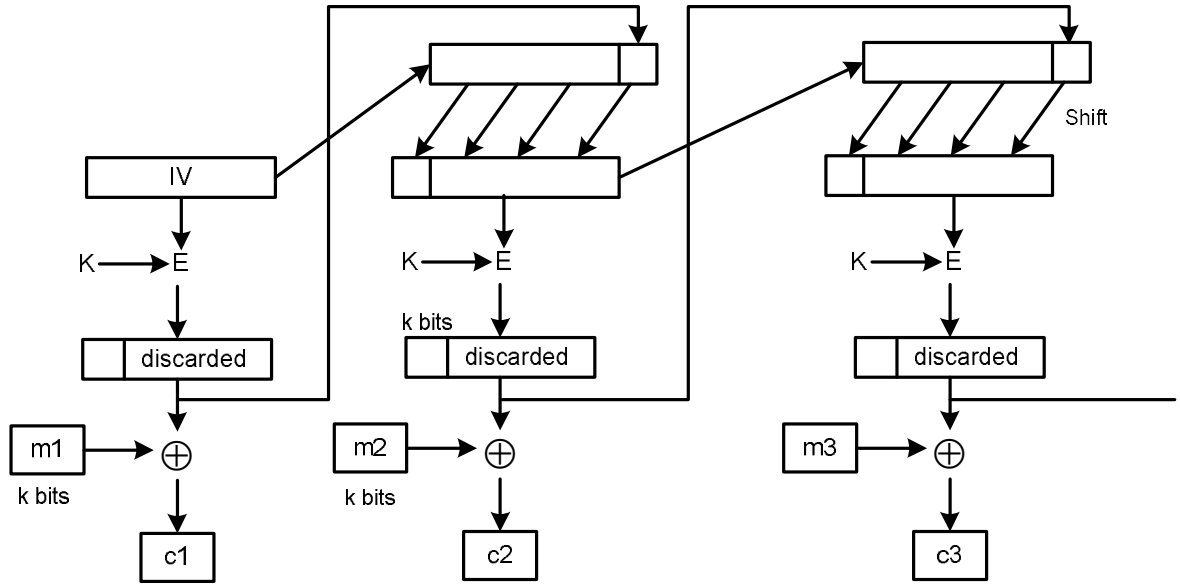


Figure 19: The output feedback mode of operation, k-bit OFB (after Kaufman et al., 2002; Stallings, 2005)

With OFB, if some bits of the ciphertext are garbled, only those bits of plaintext get garbled as well, in opposition with CBC and CBF operation modes. OFB has no error propagation. Stallings (2005) mentioned that OFB is vulnerable to message stream modification attacks. An attacker can modify some bits in the ciphertext that can have effect on the same bits in the recovered plaintext. So, attackers can make controlled changes to the recovered plaintext. With OFB mode, it is essential that the shift registers in the encryption process are identical with the shift registers for the decryption process, because otherwise the recovered plaintext is not the transmitted one (in fact, it's non sense). Schneier (1996) specified that when OFB is used, there should be also implemented a mechanism for detecting a synchronization loss and a mechanism to fill both shift registers with a new (or the same) IV to regain synchronization. The author also recommends that OFB should be used when the feedback size is the same as the block size. Another aspect refers to the generated one time pad (key stream) that is XORed with the plaintext. The same key stream should not be used with the same key, because this can be exploited by an attacker and achieve no security at all.

If, for example, two plaintext message blocks m_i and m'_i are encrypted with the same n-bit key K, then the resulting ciphertext blocks are $c_i = m_i \oplus K$ and $c'_i = m'_i \oplus k$. If the two cipher blocks are XORed, then the effect of the encryption is removed:

$c_i \oplus c'_i = (m_i \oplus K) \oplus (m'_i \oplus K)$

$= m_i \oplus m'_i \oplus K \oplus K$

$= m_i \oplus m'_i \oplus 0$

$= m_i \oplus m'_i$

If one of the plaintext is known to the adversary, then it easy to recover the other plaintext. It is worth mentioning here that block ciphers can be transformed in stream ciphers, based on the different operation modes in which they are used. For instance, if a block cipher operates in CFB mode, this leads to a non-synchronous (self-synchronizing) stream cipher. This means that the next state in generating the ciphertext depends on previously generated ciphertext units. When a block cipher operates in OFB mode, this leads to a synchronous (additive) stream cipher. This means that the next state in the encryption process does not depend on previously generated ciphertext units, and the one-time pad used in the encryption process can be first generated and then XORed with the plaintext (Oppliger, 2005).

5. Counter Mode (CTR)
   In counter mode (CTR) there is generated a one-time pad (counter), encrypted with the encryption key and then this is XORed with the plaintext unit block. For every plaintext unit block the counter has to be different. Typically, the initial counter value is further incremented by 1 for each subsequent plaintext block. There is no chaining in the counter mode. For encryption, the counter is encrypted with the cipher, and then it is XORed with the plaintext block to produce the ciphertext block. For decryption, the same sequence of counter values is used. Each encrypted counter is then XORed with the corresponding ciphertext block in order to recover the corresponding plaintext block.

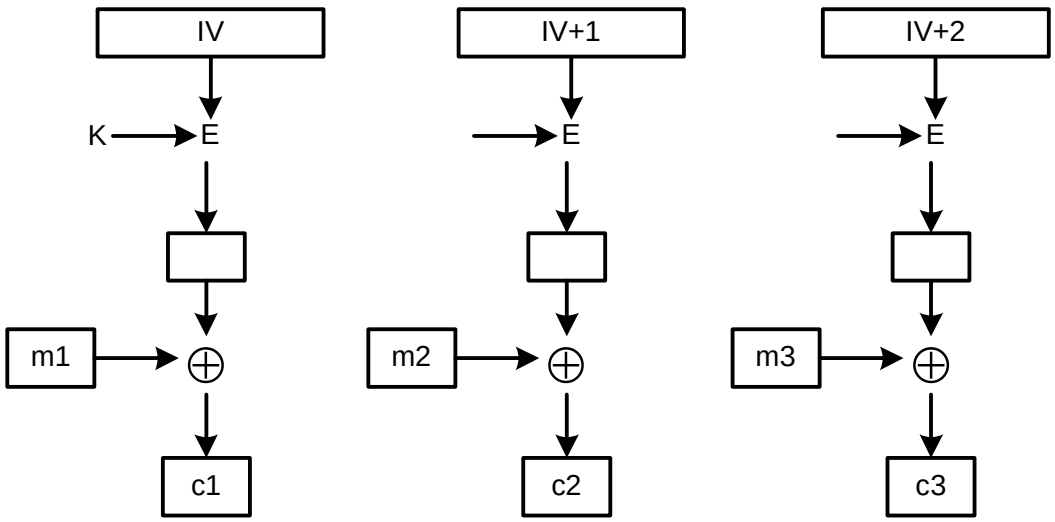


Figure 20: Counter Mode

With CTR mode, the plaintext blocks can be recovered from the ciphertext blocks in any order. As in OFB mode, there should not be used the same IV and

the same key for encrypting multiple plaintext blocks because of the security loss.

Hash functions

Message Authentication

Message authentication represents a mechanism or service used to verify the integrity of a message. Message authentication assures the integrity of the data (e.g. the data was not modified or replayed) and checks also that the identity of the sender is valid (Stallings, 2005). Oppliger (2005, p. 291) exposed the general methods implied in cryptography for authenticating messages. So, messages can be authenticated by using:

- Public-key cryptography and digital signatures, or
- Secret-key cryptography, message authentication codes (MAC) and secure hash functions.

Briefly, Stallings (2005) outlined that message authentication represents a mechanism to verify that transferred messages come from the alleged source and they have not been altered in traffic. Message authentication may also verify sequencing and timeliness.

Intro

A hash function basically maps a variable-length message into a fixed length hash value, or message digest. For achieving message authentication, a secure hash function must be combined in some fashion with a secret key. Essentially, cryptographic hash algorithms are intended to prove authenticity. The message digest resulted after a hash algorithm is applied on a data message serves as an identifying fingerprint for the respective data. So, if one bit in the input data is altered, then the hash should be completely different (Reinhold, 2005). Thus, hashing is used to prevent tampering of electronic messages. Olsen (2005) defines a hash as being a numerical code generated from a string of text when a message is sent. The receiving entity checks it against a hash it creates from the same text, and if the two resulting hashes are the same, it can be concluded that the transmitted message was not altered in traffic. The hash functions incorporate a compression function. Stallings (2005) noted that the compression function used in secure hash algorithms falls into one of two categories: a function specifically designed for the hash function or a symmetric block cipher. Moreover, most hash functions that have achieved widespread use rely on a compression function specifically designed for the hash function. A cryptographic hash function has specific properties. Below we mention the properties that are relevant from a cryptographic viewpoint.

- A hash function h is *preimage resistant* if it is computationally infeasible (hard) to find an input x with $h(x) = y$ for a given (and randomly chosen) output y.

In other words, a hash function is *preimage resistant* if, given a random hash code, it is computationally infeasible to find an input that the hash function maps to that hash code (Dent & Mitchell, 2005).

- A hash function *h* is *second-preimage resistant* or *weak collision resistant* if it is computationally infeasible to find a second input *x'* with $x' \neq x$ and $h(x') = h(x)$ for a given (and randomly chosen) input *x*.

Namely, A hash function is *second preimage resistant* if, given an input to the hash function, it is computationally infeasible to find a second input that gives the same hash code (Dent & Mitchell, 2005).

- A hash function *h* is *collision resistant* or *strong collision resistant* if it is computationally infeasible to find two inputs *x, x'* with $x' \neq x$ and $h(x') = h(x)$.

More specifically, a hash function is *collision resistant* if it is computational infeasible to find two inputs that give the same hash code (Dent & Mitchell, 2005).

According to the above definitions of the properties for hash functions, it results that collision resistant hash function must be second-preimage resistant. Oppliger (2005) pointed out that otherwise it would be possible to find a second preimage for an arbitrary chosen input, and this second preimage would yield a collision. On the other hand, a second-preimage resistant hash function must not be collision resistant. Resultantly, collision resistance implies second-preimage resistance, but not the opposite. Based on the properties enumerated above, there could be defined one-way hash functions (OWHF) and collision resistant hash functions (CRHF). A one-way hash function is a hash function that is preimage resistant and second-preimage resistant. A collision resistant hash function is a hash function that is preimage resistant and collision resistant. It is worth mentioning that a CRHF is always an OWHF, while the opposite might not be true. RSA Security, Inc. played an important role in the development and deployment of many practically relevant cryptographic hash functions (Oppliger, 2005). RSA Security, Inc. firstly designed a proprietary hash function named MD (message digest). The first published hash function is MD2[1] that was widely used, mainly in the secure messaging products of RSA Security, Inc. Next, the same company designed MD4[2] that is specified in RFC 1320. Because of some reported weaknesses of MD4, a new hash function was designed – MD5[3] by Ronald Rivest and is specified in RFC 1321. Although, MD5 was considered to be more secure than MD4 it is also a little bit slower (Oppliger, 2005). In 1993, the US National Institute of Standards and Technology (NIST) designed the Secure Hash Algorithm (SHA). Oppliger (2005) noted that SHA algorithm is similar to MD5, but even more strengthened and also a little bit slower. Next, NIST revised the initial SHA version and released a new version SHA-1 specified in the Federal Information Processing Standards Publication (FIPS PUB) 180-1[4]. In 2002, NIST performed a revision of the standard, and this resulted in a new FIPS PUB 180-2[5] that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512 bits. Finally, the NIST secure hash standard contains the specifications for five hash functions: SHA-1, SHA-224, SHA-256, SHA-384, and SHA-

[1] http://www.faqs.org/rfcs/rfc1319.html
[2] http://www.faqs.org/rfcs/rfc1320.html
[3] http://www.faqs.org/rfcs/rfc1321.html
[4] http://www.itl.nist.gov/fipspubs/fip180-1.htm
[5] http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

512. MD4 and MD5 produce hash values of 128 bits, while SHA-1 produces hash values of 160 bits. The revised versions of SHA produce hash values that are even longer. Oppliger (2005) acknowledged that from a security viewpoint, long hash values are given preference due to the fact that the likelihood of collisions is reduced. Thus, it is advisable to replace MD5 with SHA-1 or any other function from the SHA family where possible. Next, in this sub-chapter we discuss the security of MD5 and SHA hash functions.

MD5

As we have already mentioned, MD5 is a strengthened version of MD4. MD5 algorithm takes an input message of arbitrary length and produces a 128-bit hash value of the message. The input message is processed in 512-bit blocks which can be divided into 16 32-bit sub-blocks. The message digest is a set of 4 32-bit blocks that concatenate in order to form a single 128-bit hash code. Cryptographic hash functions are typically attacked by trying to find two different documents that have the same hash value (a collision attack). Through collision attacks, the security of hashes is defeated. It has been shown in the literature (Kaufman et al., 2002; Oppliger, 2005; Reinhold, 2005) that based on the mathematical result known as the "Birthday Paradox", one needs to try 2 to N/2 possibilities, for finding a collision (N represents the number of bits in the output). So, MD5 hash function that has a 128 bits output provides only 64 bits of strength against brute force attacks. A series of collision attacks have been reported in the literature[1] against MD5. Thus, Schneier (2005) concluded on his weblog[2] that MD5 is broken.

SHA-1

Oppliger (2005) noted that SHA-1 hash function is conceptually and structurally similar to MD4 and MD5. But, SHA-1 outputs hash values of 160 bits. So, taking into consideration the "Birthday Attack", SHA-1 only provides 80 bits of strength, at most. In 2005 NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010. Although SHA-1 was considered secure, shortly after NIST's decision to replace SHA-1, a research team (Wang et al., 2005) succeeded to break this hash function. In their paper, the researchers presented new collision search attacks on the hash function SHA-1. They showed that collisions of SHA-1 can be found with complexity less than 2^{69} hash operations. This represents the first attack on the full 80-step SHA-1 that has complexity less than the 2^{80} theoretical bound as we have previously mentioned for a 160 bits hash function. Later, in August 2005, another attack on SHA-1 was reported[3] about finding collisions for SHA-1 in 2^{63} operations. Stallinger (2005) concluded that these results should accelerate the transition to the other versions of SHA. These findings, regarding the collision attacks, are of interest for applications that require collision resistant hash functions. Anyway, as Reinhold (2005) suggested, "there is no need to panic". Taking into consideration the results of the researches about collision attacks that showed that SHA-1 has at most 63 bits strength, this is still non-trivial to perform such an attack with the current technology. It is recommended anyway to develop upgrade plans for adopting stronger

[1] http://www.cits.rub.de/MD5Collisions/ ; http://www.schneier.com/blog/archives/2005/06/more_md5_collis.html; http://eprint.iacr.org/2006/104.pdf accessed May 2007

[2] http://www.schneier.com/blog/archives/2005/08/the_md5_defense.html accessed May 2007

[3] http://www.rsa.com/rsalabs/node.asp?id=2927 accessed May 2007

cryptographic hash functions. Reinhold (2005) and Szydlo (2005) suggested viable approaches for improving the security of applications:

- SHA-1 should not be used in new designs. It should be replaced with stronger variants. For instance, SHA256, SHA384 and SHA512 are widely available and free. In any case, MD5 should be phased out because it is even weaker than SHA-1, but still widely used.
- Another suggestion is to add randomness to hash functions. But in order to implement this, the applications must have a good source of randomness and should also change the protocols that use hash functions.

Next, we provide a comparison of SHA parameters for hash functions. In this table, all sizes are measured in bits, and the security refers to the fact that a "Birthday Attack" on a message digest of size N produces a collision with a probability of approximately $2^{N/2}$. (after Stallings, 2005)

	SHA-1	SHA-256	SHA-384	SHA-512
Message Digest Size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80
Security	80	128	192	256

Message authentication codes

A message authentication code (MAC) is an algorithm that requires the use of a secret key. So, a secret-key algorithm can be used to generate a small fixed-size block of data, known as a cryptographic checksum or MAC. The MAC is computed on a message with a secret key, and verified with the same secret key by the receiver (Oppliger, 2005). Consequently, MAC depends on both the message it authenticates and the secret key that only the legitimate sender and the legitimate recipient(s) are assumed to know. It is worth noting that there is a fundamental difference between message authentication using MACs and message authentication using digital signatures. For MACs, the same secret key is used for computation of the MAC value and for verifying it, while for digital signatures the secret key is used for signing a message, while the public key is used for verifying the signature. Digital signatures can be used for achieving also non-repudiation services, whereas MACs are used only for integrity purposes. Another difference between MACs and digital signatures is that MACs can be verified only by the legitimate possessors of the secret key, while the digital signatures can be verified by any entity that is in the possession of the public key.

Computationally Secure MACs

Oppliger (2005, p. 294) presented some possibilities to design MACs that are computationally secure. Some of these possibilities are:

- MACs that use symmetric encryption systems
- MACs that use keyed hash functions
- MACs that use pseudorandom functions (PRFs)
- MACs that use universal hash functions

Public Key Infrastructure (PKI)

A public key infrastructure (PKI) consists of all the elements (certificates, a repository for retrieving certificates, a method of revoking certificates, a method of evaluating a chain of certificates) necessary to securely distribute public keys. In fact, a certificate is a message vouching that a certain name goes with a corresponding public key (Kaufman et al., 2002). There are three types of certificates: end-user certificates, CA certificates and cross-certificates. Certificates are issued by certifications authorities (CAs). The CAs is a trusted company universally or for a certain domain depending on the PKI Trust Model that is used (e.g. monopoly, oligarchy, anarchy PKI trust models). CAs posses PKI trust anchors (a public key that has been previously verified and that is trusted to sign certificates) that are used for issuing certificates to other entities. The monopoly PKI trust model is not realistic due to the difficulties of using only one organization for checking the credentials of the entities that request certificates, of issuing these certificates, and providing further validation and revocation services. A model that is typically implemented in browsers is the oligarchy PKI trust model in which the different entities and products are configured and issued certificates by different CAs that have trust anchors. Certificates can be used in a variety of situation within protocols and in secure communications. For instance, if an entity *A* wants to securely find the public key of another entity *B* for sending an him/her an encrypted message, then *A* can use a PKI Trust Model for retrieving the public key of *B*. Also, by the means of certificates, the identity, the digital signature, the public key of target entities can be verified and validated (If entity *A* wants to find a path to *B*'s key, then the name of *B* is the *target*). Nevertheless, due to the scope of this book, we propose that the PKI should be considered for future research in the context of Jericho Project.

Comparison of cryptographic primitives

In the field of cryptography new algorithms surface continuously and existing algorithms are continuously attacked. Thus, many algorithms that were believed to be strong against attacks were demonstrated to be weak in front of new designed attacks. In this sub-chapter we present a comparison of some of the most cryptographic algorithms and primitives investigated in this book. The cryptographic algorithms provide different levels of cryptographic and security strength, depending on the algorithm itself and on the variety of key sizes that are used. The security strength of a cryptographic algorithm for a given key size is traditionally described in terms of the amount of effort it takes to break it. Stallings presented a report of Certicom[1] containing the comparison of different cryptographic algorithms and primitives in terms of computational effort for cryptanalysis for comparable key sizes.

[1] http://www.certicom.com/ accessed June 2007

Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis (after Certicom):

Symmetric Scheme (key size in bits)	**ECC-Based Scheme (size of *n* in bits)**
56	112
80	160
112	224
128	256
92	384
256	521

We observe that computational effort for breaking ECC-based schemes is comparable with the effort of breaking symmetric cryptographic algorithms. This is due to the fact that ECC implementations are smaller and more efficient than the implementation of other public-key algorithms. U.S. National Security Agency (NSA) presented a report[1] of NIST in which the conventional cryptographic algorithms are compared with ECC.

NIST Recommended Key Sizes:

Bits of security	**Symmetric-key algorithms**	**Hash algorithms**	**RSA and Diffie-Hellman Key Size (bits)**	**Elliptic Curve Key Size (bits)**
80		SHA-1	1024	160
112	3DES		2048	224
128	AES-128	SHA-256	3072	256
192	AES-192	SHA-384	7680	384
256	AES-256	SHA-512	15360	521

As presented in this Table, for using RSA or Diffie-Hellman to protect 128-bit AES keys, there should be used 3072-bit keys. Based on the results presented by NIST, the equivalent key size for ECC is 256 bits. As the symmetric key size increases, the required key size for RSA and Diffie-Hellman increases at a considerably faster rate than the required key for ECC. Thus, in the NSA2 report it is stated that ECC offers more security per bit increase in key size than RSA or Diffie-Hellman public-key cryptographic algorithms. The ECC security increases more rapidly as key length increases. In the same report there are presented results that illustrate the fact that ECC is computationally more efficient than RSA and Diffie-Hillman algorithms for the same symmetric key size. ECC' mathematical foundation is more complex than either RSA or DH arithmetic, but ECC offers added strength per bit.

[1] http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm accessed June 2007
[2] http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm accessed June 2007

Relative Computation Costs of Diffie-Hellman and Elliptic Curves[1]. This table shows the ratio of DH computation versus EC computation for each of the key sizes listed in Table on page 134.

Symmetric Key Size (bits)	**Ratio of DH Cost : EC Cost**
80	3:1
112	6:1
128	10:1
192	32:1
256	64:1

For protecting both classified and unclassified National Security information, the National Security Agency has decided to move to elliptic curve based public key cryptography. Where appropriate, NSA plans to use the elliptic curves over finite fields with large prime moduli (256, 384, and 521 bits) published by NIST. In the same report of NSA it is stated that the United States, the UK, Canada and certain other NATO nations have all adopted some form of elliptic curve cryptography for future systems to protect classified information throughout and between their governments. NIST[2] presented as even a more detailed comparison of the equivalent cryptographic algorithms strength and made the following recommendations regarding the cryptographic algorithms to be used further for achieving secure communications and the minimum key sizes.

Recommended algorithms and minimum key sizes by NIST[3]

[1] http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm accessed June 2007
[2] http://csrc.nist.gov/encryption/kms/guideline-overview%20(b-w).pdf accessed June 2007
[3] http://csrc.nist.gov/CryptoToolkit/kms/SP800-57Part1August2005.pdf accessed April 2007; http://www.keylength.com/en/4/ accessed June 2007

Algorithm security lifetimes	Symmetric key algorithms	RSA & D-H	ECC	Hash (A)	Hash (B)
2007 to 2010 in. of 80 bits of strength	2TDEA 3TDEA AES-128 AES-192 AES-256	1024	160	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
2011 to 2030 (min. of 112 bits of strength)	3TDEA AES-128 AES-192 AES-256	2048	224	SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
Beyond 2030 (min. of 128 bits of strength)	AES-128 AES-192 AES-256	3072	256	SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
>> 2030 (min. of 192 bits of strength)	AES-256	7680	384	SHA-384 SHA-512	SHA-224 SHA-256 SHA-384 SHA-512
>>> 2030 (min. of 256 bits of strength)		15360	512	SHA-512	SHA-256 SHA-384 SHA-512

A few observations regarding the recommended algorithms and minimum key sizes:

- Hash (A): are used for digital signatures and hash-only applications
- Hash (B): are used for HMAC, Key Derivation Functions and Random Number Generation
- SHA-1 has recently been demonstrated to provide less than 80 bits of security for digital signatures; the security strength against collisions is assessed at 69 bits. The use of SHA-1 is not recommended for the generation of digital signatures in new systems; new systems should use one of the larger hash functions. For the present time, SHA-1 is included here to reflect its widespread use in existing systems, for which the reduced security strength may not be of great concern when only 80-bits of security are required.

The algorithms and key sizes in the table are considered appropriate for the protection of data during the given time periods. Algorithms or key sizes not indicated for a given range of years shall not be used to protect information during that time period. This NIST Recommendation applies to U.S. government agencies using cryptography for the protection of their sensitive unclassified information. However, this recommendation may also be followed, on a voluntary basis, by other organizations that want to implement sound security principles in their computer systems. This recommendation advises the users of cryptographic mechanisms on the appropriate choices of

algorithms and key sizes. As NIST specified algorithm suites that combine non-comparable strength algorithms are generally discouraged. However, algorithms of different strengths and key sizes may be used together for performance, availability or interoperability reasons, provided that sufficient protection is provided. Generally, the weakest algorithm and key size used to provide cryptographic protection determines the strength of the protection. For instance, when security protocols are used for achieving the requirements for secure communications in Jericho networks, determination of the strength of protection provided for information includes an analysis not only of the algorithm(s) and key size(s) used to apply the cryptographic protection to the information for achieving confidentiality, but also any algorithm(s) and key size(s) associated with establishing the key(s) used for information protection itself.

NIST recommends the following configuration[1] of algorithms and key sizes for securing transferred data with confidentiality, integrity, authentication and non-repudiation protection:

- Confidentiality: Encrypt the information using AES-128. Other AES key sizes would also be appropriate, but perform a bit slower. In addition, another block cipher could be used for achieving confidentiality for the transmitted data, namely Camellia cipher.

  Camellia is comparable with AES in terms of security and performance. It was also adopted into various standard/recommended specifications. As a result, Camellia is adopted in security protocols, such as SSL/TLS, IPsec, XML etc.

- Integrity, authentication and non-repudiation: It is supposed that only one cryptographic operation is preferred; for instance digital signatures. SHA-256 could be selected for the hash function. An algorithm for digital signatures should be selected from what is available to an application (e.g. ECDSA with at least a 256-bit key).

The Committee on National Security Systems2 recommended that the design and strength of all key lengths of the AES algorithm (e.g. 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths.

European Network of Excellence for Cryptology recommended slightly larger key sizes. Recommended algorithms and minimum key sizes by ECRYPT 2007[3]

[1] http://csrc.nist.gov/CryptoToolkit/kms/SP800-57Part1August2005.pdf accessed June 2007
[2] www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf accessed June 2007
[3] http://www.keylength.com/en/3/ accessed April 2007

Level	Protection	Symmetric	Asymmetric	Elliptic Curve	Hash
1	Attacks in "real-time" by individuals *Only acceptable for authentication tag size*	32	-	-	-
2	Very short-term protection against small organizations *Should not be used for confidentiality in new systems*	64	816	128	128
3	Short-term protection against medium organizations, medium-term protection against small organizations	72	1004	144	144
4	Very short-term protection against agencies, long-term protection against small organizations *Smallest general-purpose level, protection from 2007 to 2010*	80	1248	160	160
5	Legacy standard level *Use of 2-key 3DES restricted to 106 plaintext/ciphertexts, protection from 2007 to 2016*	96	1777	192	192
6	Medium-term protection *protection from 2007 to 2026*	112	2432	224	224
7	Long-term protection *Generic application-independent recommendation, protection from 2007 to 2036*	128	3248	256	256
8	"Foreseeable future" *Good protection against quantum computers*	256	15424	512	512

Regarding the recommendations made in this table, we summarize the following recommendations:

- the 32 and 64-bit levels should not be used for confidentiality protection; 32-bit keys offer no confidentiality at all relative to any attacker, and 64-bit offers only very poor protection.
- while both 80 and 128-bit keys provide sufficient security against brute force key-search attacks (on symmetric primitives) by the most reasonable adversaries, it should be noted that 80 bits would be practically breakable and 128 bits might correspond to an effective 80-bit level, if one considers attack models based on pre-computation and large amounts of available storage. As a simple rule of thumb, one may choose to double the key size to mitigate threats from such attacks.

- the main consideration for a secure hash function is the size of the outputs. If the application requires collisions to be difficult to find, the output must be twice the desired security level. This is the case when used with digital signatures for instance. When used as a keyed hash for message authentication, however, the outputs may often be truncated.
- As a remark, 256-bit symmetric key offers good protection against quantum computers.

Stallings (2005) mentioned that the principal attraction of ECC, compared to RSA, is that it appears to offer equal security for a far smaller key size. Consequently, the process overhead is reduced. On the other hand, the theory of ECC appeared in the last decennia and it was not subject to sustained cryptanalytic analysis for finding its weaknesses. So, ECC was less researched than the others cryptographic algorithms and this why it is not yet recommendable to use it for securing the transferred information. Accordingly, the confidence level in ECC is not yet as high as that in RSA. When there are conventional cryptographic algorithms available that offer the same level of protection as ECC, even at the cost of computational resources, these should be preferred for usage in designing secure protocols instead of using ECC in the context of Jericho Project. However, the choice of the cryptographic algorithms to be used for designing security protocols in the context of Jericho Project depends on the security needs of the data, on the classification type of the data and on its security level. Also, if an increased number of transfers of sensitive data over Internet occurs, the mechanisms offered by ECC can be used within the security protocols proposed for securing the communications for Jericho Project.

5. Conclusions and Future Directions for Research

In the context of Jericho Forum the need for securing data in transit over the Internet is essential due to the new threats that occur and the new ways business are nowadays conducted in a collaborative environment. Over the past decennia, cryptography has become the pillar for providing securing communications over the Internet. Cryptography provides the security mechanisms needed for accomplishing the security services desired for secure communications.

The aim of this book was to investigate and recommend the most appropriate security mechanisms offered by cryptography for being used in security protocols that offer protection for the data transmitted over the Internet in the contextual framework of Jericho Project. In the course of this research we analyzed and made recommendations regarding the security requirements, the security protocols and the cryptographic algorithms that could be used in the context of Jericho Project for achieving end-to-end security. In essence, cryptographic algorithms, primitives and protocols are the means of designing and deploying secure communications in Jericho networks as well. Cryptography is at the basis of IPsec, SSL/TLS and XML Encryption protocols that we have analyzed in this book. Consequently, in this book we have investigated the means offered by cryptography for designing secure protocols in order to obtain end-to-end security in Jericho networks. An essential aspect regarding the cryptographic algorithms and primitives used to design secure protocols in the context of Jericho Project, is that some cryptographic algorithms provide different levels of cryptographic strength, and implicitly security, through a variety of key sizes. Evidently, the cryptographic algorithms may be combined in many ways to support the design of secure protocols, but we have to select the most adequate means offered by cryptography for acquiring the goals of secure communications in Jericho Project. In the course of our research we concluded that the security of information in transit over the Internet, which is protected by cryptographic mechanisms within secure protocols, depends on the strength of the cryptographic keys, the effectiveness of mechanisms and protocols associated with keys, and the implementation of the protocols for the adequate situations. Nevertheless, protecting the information in traffic is not solely dependent on the mathematical strength of the chosen cryptographic algorithms to be used for designing security protocols. Besides the level of security attached to the data as a result of the classification process, there are also other factors that are taken into consideration when deciding which cryptographic mechanisms to deploy for achieving the requirements for secure communications in Jericho Project. If more than one algorithm and key size is available, the selection may be based on algorithm performance, memory requirements, as long as the minimum requirements are met. However, this research does not address implementation details for the cryptographic algorithms and security protocols that can be used in Jericho networks for achieving the security requirements identified for secure communications.

Users and developers have many choices in their use of cryptographic mechanisms for designing security protocols in the context of Jericho Project. Designing pervasive, inherently secure protocols for achieving secure communications for Jericho networks is

not an easy task. There are more ways to design a security protocol for achieving the goals for secure communications in Jericho networks. As stated by Jericho Forum Commandments, the protocols should be used open, secure, and flexible. Moreover, the security mechanisms employed for the accomplishment of the security services required for secure communications in Jericho Project must be pervasive, simple, scalable and easy to manage. However, Cole et al. (2005) remarked that while cryptography can be very secure when used properly, the human element of the process should always be considered and taken into consideration. So, making the users aware of how to protect the privacy and integrity of the business and personal data against the different threats is one of the first steps when implementing a security policy based on Jericho commandments.

Future Directions for Research

In the course of our research we determined some interesting topics for future investigation in the context of Jericho Project. We consider the following topics of interest for further analysis and investigation within the purpose of achieving secure communications in Jericho networks:

- Different methods of authentication of the entities within the security protocols for secure communications

  Although we have investigated this topic in this book, we acknowledge that it can be more in depth analysed, and recommendations regarding the most appropriate solutions for authentications should be made in the context of Jericho Project.

- The relationship of the Trust Broker with the Certification Authorities represents another research topic that deserves more attention in a future investigation

- Key Management provides the foundation for the secure generation, storage, distribution, and destruction of keys. In the context of Jericho Project, this subject should be further tackled, and the best approaches for protecting the secret and private keys against unauthorized disclosure should be recommended.

- Implementation details of the cryptographic algorithms and security protocols recommended for securing the communications in the context of Jericho Project

- Mechanisms for securing the data in storage and in process should be also further investigated in the context of Jericho Project

- Other security mechanisms and protocols for securing Web Services

- The possibilities offered by the Authenticated Encryption systems in the context of Jericho Project. Authenticated Encryption refers to encryption systems that simultaneously protect confidentiality and authenticity (integrity) of communications.

References

Alshamsi, A. & Saito, T. (2005). A Technical Comparison of IPSec and SSL. Proceedings of the 19th International Conference on Advanced Information Networking and Applications - Volume 2

Andress, A. (2004).Surviving Security: How to Integrate People, Process, and Technology, Second Edition. Auerbach Publications

Atreya, M., Hammond, B., Paine, S., Starrett, P., & Wu, S. (2002). Digital Signatures. McGraw-Hill/Osborne

Bernstein, D. J. (2005).Cache-timing attacks on AES. The University of Illinois at Chicago available at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

Biham, E. & Shamir, A. (1990). Differential cryptanalysis of DES-like cryptosystems. The Weizmann Institute of Science Department of Applied Mathematics

Biham, E., Dunkelman, O., & Keller, N. (2005). Related-Key Boomerang and Rectangle Attacks. Advances in Cryptology – EUROCRYPT 2005, LNCS 3494, Springer, p. 507-525

Blaze, M., Diffie, W., Rivest, R.L., Schneier, B., Shimomura, T., Thompson, E., Wiener, M. (1996). Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. A Report by an Ad Hoc Group of Cryptographers and Computer Scientists

Bragg, R. (2004). MCSE Self-Paced Training Kit (Exam 70-298): Designing Security for a Microsoft Windows Server 2003 Network. Microsoft Press

Burnett, M.M. & Foster, J.C. (2004). Hacking the Code: ASP.NET Web Application Security. Syngress Publishing

Canavan, J.E. (2001). Fundamentals of Network Security. Artech House, Inc.

Cheon, J.H., Kim, M., Kim, K., Lee, J.Y., & Kang, S. W. (2001). Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. Information Security and Cryptology - ICISC 2001

Cole, E, Krutz, R.L., & Conley, J.W. (2005). Network Security Bible. Wiley Publishing, Inc.

Conrad, E.(2007). Types of Cryptographic Attacks. available at http://www.giac.org/resources/whitepaper/cryptography/57.php accessed May 2007

Dam, K.W. & Lin, H.S. *Editors* (1996). Cryptography's Role in Securing the Information Society. Committee to Study National Cryptography Policy, Computer Science and Telecommunications Board, National Research Council, Washington, D.C.

Daemen, J., Govaerts, R. & Vandewalle, J. (1994). Weak keys of IDEA. Advances in Cryptology, Proceedings Crypto'93, LNCS 773, D. Stinson, Ed., Springer-Verlag, p. 224-231

De Laet, G. & Schauwers, G. (2004). Network Security Fundamentals. Cisco Press

Demirci, H., Selçuk, A. A., & Türe, E. (2003). A New Meet-in-the-Middle Attack on the IDEA Block Cipher. 10th Annual International Workshop, Selected Areas on Cryptography, p. 117-129

Dent, A.W. & Mitchell, C.J. (2005). User's Guide to Cryptography and Standards. Artech House Computer Security Series

DES Modes of Operation (1981). FIPS PUB 81, National Bureau of Standards, U.S. Department of Commerce

Diffie, W. & Hellman, M.E. (1976). New Directions in Cryptography. IEEE Transactions on Information Theory, v. IT–22, n.6, p. 644–654

:Graw-Hill/Osborne

e XML: The New Syntax for Signatures and Encryption. Addison Wesley

El Aoufi, S. (2006). Cryptographie en ICT 2e druk Theorie en Praktijk. Academic Service Sdu Uitgevers BV

Erl, T. (2004).Service-Oriented Architecture A Field Guide to Integrating XML and Web Services. Prentice Hall PTR

.

Feistel, H. (1973). Cryptography and Computer Privacy. *Scientific American*, 228(5), 15-23

Ferguson, N. & Schneier, B. (1999). A Cryptographic Evaluation of IPsec White Paper. Counterpane Internet Security, Inc., available at www.schneier.com/paper-ipsec.pdf

Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Wagner, D., Whiting, D. (2000). Improved cryptanalysis of Rijndael. In Bruce Schneier, editor, Proceedings of Fast Software Encryption – FSE'00, number 1978 in Lecture Notes in Computer Science, pages 213–230. Springer-Verlag, 2000

Galbreath, N. (2002). Cryptography for Internet and Database Applications Developing Secret and Public Key Techniques with Java. Wiley Publishing, Inc.

Gilbert, H. & Minier, M. (2000). Acollision attack on seven rounds of Rijndael. In Proceedings of the Third Advanced Encryption Standard Conference, p. 230– 241, NIST

Goots, N., Izotov, B., Moldovyan, A., & Moldovyan, N. (2003). Modern Cryptography: Protect Your Data with Fast Block Ciphers. A-LIST Publishing

Gregg, M. (2006). Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network. Syngress Publishing

Gutmann, P. available at http://www.cs.auckland.ac.nz/~pgut001/tutorial/index.html accessed April 2007

Harris, S. (2005). CISSP: All-in-One Exam Guide, Third Edition. McGraw-Hill/Osborne

Hartman, B., Flinn, D. J., Beznosov, K., & Kawamoto, S. (2003). Mastering Web Services Security. Wiley Publishing, Inc.

Hassler, V. (2001). Security Fundamentals for E-Commerce. Artech House, Inc.

Hershey, J. E. (2003). Cryptography Demystified. McGraw-Hill

Howard, M. & Lipner, S. (2006). The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software. Microsoft Press

Jakimoski, G. & Desmedt, Y. (2004).Related-Key Differential Cryptanalysis of 192-bit Key AES Variants. SAC 2003, LNCS Vol. 3006, p. 208-221, Springer

Jaworski, J. & Perrone, P.J. (2000). Java Security Handbook. Sams Publishing

Jericho Forum (2006). Position Paper Federated Identity. Jericho Forum Publications available at https://www.opengroup.org/jericho/publications.htm

Jericho Forum (2006). Position Paper End Point Security. Jericho Forum Publications available at https://www.opengroup.org/jericho/publications.htm

Jericho Forum (2006). Jericho Forum Commandments. Jericho Forum Publications Version 1.1 December 2006, available at www.opengroup.org/jericho/commandments_v1.1.pdf

Jericho Forum (2007). White Paper – Business rationale for de-perimeterization. Jericho Forum Publications available at http://www.opengroup.org/jericho/Business_Case_for_DP_v1.0.pdf

Jericho Forum (2007). Position Paper – Principles for Managing Data Privacy. Jericho Forum Publications available at http://www.opengroup.org/jericho/Privacy_v1.0.pdf

Johnston, A.B. & Piscitello, D.M. (2006). Understanding Voice over IP Security. Artech House

Kaliski, B. (2000). Requirements for New Public-Key Algorithms. RSA Laboratories

Kaufman, C., Perlman, R. & Speciner, M. (2002). Network Security Private Communication in a Public World. Prentice Hall PTR

Kocher, P., Jaffe, J., Jun, B., Laren, C., & Lawson, N. (2002-2003). Self-Protecting Digital Content A Technical Report from the Cri Content Security Research Initiative. Cryptography Research, Inc. (CRI)

Komar, B. & the Microsoft PKI Team (2004). Microsoft Windows Server 2003 PKI and Certificate Security. Microsoft Press

Kruegel, C. (2005). Internet Security. The Industrial Communication Technology Handbook, CRC

Press
Lail, B.M. (2002). Broadband Network & Device Security. McGraw-Hill/Osborne
Larson, R. E. & Cockcroft, L. (2003). CCSP: Cisco Certified Security Professional Certification All-in-One Exam Guide. McGraw-Hill/Osborne
Lehtinen, R. (2006). Computer Security Basics, 2nd Edition. O'Reilly
Lucks, S. (2000). Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In Proceedings of the Third Advanced Encryption Standard Conference. NIST
Mallery, J. et al. (2005). Hardening Network Security. McGraw-Hill/Osborne
Mao, W. (2003). Modern Cryptography: Theory and Practice. Prentice Hall PTR
Mar-Elia, D., Melber, D., Stanek, W. & Microsoft Group Policy Team (2005). Microsoft Windows Group Policy Guide. Microsoft Press
Menezes, A.J., van Oorschot, P.C., & Vanstone, S. A. (1996).Handbook of Applied Cryptography. CRC Press
Merkow, M.S. CCP, & Breithaupt, J. (2000). The Complete Guide to Internet Security. AMACOM
Microsoft Corporation (2005). Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0. Microsoft Press
Microsoft Corporation (2003). Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication. Microsoft Press
Microsoft Corporation (2003). Improving Web Application Security: Threats and Countermeasures. Microsoft Press
Miller D. (2005). Black Hat Physical Device Security: Exploiting Hardware and Software. Syngress Publishing
Nantz, B. & Moroney, L. (2005). Expert Web Services Security in the .NET Platform. Apress
Nash, A., Duane, W., Joseph, C., Brink, D. (2001). PKI: Implementing and Managing E-Security. The McGraw-Hill Companies
Northrup, T. & Thomas, O. (2004). MCSA/MCSE Self-Paced Training Kit (Exam 70-299): Implementing and Administering Security in a Microsoft Windows Server 2003 Network. Microsoft Press
O'Neill, M. et al. (2003). Web Services Security. McGraw-Hill/Osborne
Olifer, N. & Olifer, V. (2006). Computer Networks: Principles, Technologies and Protocols for Network Design. John Wiley & Sons Inc.
Olsen, F. (2005). Hashing out encryption. available at http://www.fcw.com/fcw/articles/2005/0207/web-hash-02-07-05.asp
Oppliger, R. (2002). Internet and Intranet Security, Second Edition. Artech House, Inc.
Oppliger, R. (2003). Security Technologies for the World Wide Web, Second Edition. Artech House, Inc.
Oppliger, R. (2005). Contemporary Cryptography. Artech House, Inc.
Osvik, D.A., Shamir, A. & Tromer, E. (2005). Cache Attacks and Countermeasures: the Case of AES. Proceedings RSA Conference Cryptographers Track (CT-RSA) 2006, Springer, 2006 availble at http://theory.csail.mit.edu/~tromer/papers/cache.pdf
Oswald, E., Daemen, J., & Rijmen, V. (2002). AES - The State of the Art of Rijndael's Security. availble at http://www.iaik.tugraz.at/aboutus/people/oswald/papers/aes_report.pdf
Papadimitratos, P. & Haas, Z. J. (2003). Secure message transmission in mobile ad hoc networks. *Ad Hoc Networks*, 1(1), 193-209
Piper, F. & Murphy, S. (2002). Cryptography: A Very Short Introduction. Oxford University Press
Poddar, V., Singh, V.K., Vinoo, A. E., Saraswat, P. (2003). Cryptography Protocols and Algorithms. SkillSoft Press
Pujolle, G. (ed) (2007). Management, Control and Evolution of IP Networks. International Scientific and Technical Encyclopedia

Quiggle, A. (2001). Implementing Cisco VPNs : A Hands-On Guide. McGraw-Hill/Osborne
Raina, K. (2003). PKI Security Solutionsfor the Enterprise: Solving HIPAA, E-Paper Act, and Other Compliance Issues. Wiley Publishing, Inc.
Ramachandran, J. (2002). Designing Security Architecture Solutions. John Wiley & Sons
Rappa, M. available at http://digitalenterprise.org/models/models.html accessed March 2007
Reinhold, A. The Attack On SHA-1 And Its Implications. available at http://www.hurwitz.com/index.php?option=com_content&task=view&id=96&Item
Reuvid, J. (2006). The Secure Online Business Handbook—A Practical Guide to Risk Management and Business Continuity, 4th Edition. Kogan Page
Rhee, M. J. (2003).Internet Security Cryptographic Principles, Algorithms and Protocols. John Wiley & Sons Ltd
Richards, R. (2006). Pro PHP XML and Web Services. Apress
Rosenberg, J. & Remy, D.L. (2004). Securing Web Services with WS-Security. Sams Publishing
Sharma, R. K. et al. (2002). Cisco Security Bible. John Wiley & Sons
Schein, P.G. (2000). MCSE Windows 2000 Security Design Exam Cram (Exam 70-220). The Coriolis Group
Schneier, B. (1996). Applied Cryptography. Second Edition, John Wiley & Sons, Inc.
Schneier, B. (2005). AES Timing Attack. Schneier on Security Blog, available at http://www.schneier.com/blog/
Seys, S. (2006). Cryptographic Algorithms and Protocols for Security and Privacy in Wireless Ad Hoc Networks. Catholic University Leuven
Simpson, P. (2006). InsidersChoice to MCP/MCSE Exam 70-293 Windows Server 2003 Certification: Planning and Maintaining a Microsoft Windows Server 2003 Network Infrastructure, Second Edition. TotalRecall Press
Slone, S. & The Open Group Identity Management Work Area (2004). Identity Management White Paper. The Open Group
Sluiter, J. (2006). Point of View Service Oriented Architecture and Deperimeterisation. Capgemini.
Smith, C. (2006). Pro Open Source Mail: Building an Enterprise Mail Solution. Apress
Snader, J.C. (2005). VPNs Illustrated: Tunnels, VPNs, and IPsec. Addison Wesley Professional
Solomon, M.G. & Chapple, M. (2005). Information Security Illuminated. Jones and Bartlett Publishers
Stallings, W. (2005). Cryptography and Network Security Principles and Practices, Fourth Edition. Prentice Hall
Stallings, W. (2003). Network Security Essentials *Applications and Standards*, Second Edition. Prentice Hall
Stamp, P., Whiteley, R., Koetzle, L., & Rasmussen, M. (2005). Jericho Forum Looks To Bring Network Walls Tumbling Down. Analyst Reports at Forrester available at http://www.csoonline.com/analyst/report3839.html
Stamp, M. (2006).Information Security Principles and Practice. JohnWiley & Sons, Inc.
Stinson, D. (1995). Cryptography: Theory and Practice. CRC Press LLC
Sullivan, D. (2006). The short guide to protecting business internet usage. Realtime publishers, available at http://nexus.realtimepublishers.com/ accessed May 2007
Szydlo, M. (2005). SHA-1 Collisions can be Found in 2^{63} Operations. available at http://www.rsa.com/rsalabs/node.asp?id=2927
Talbot, J. & Welsh, D. (2006). Complexity and Cryptography An Introduction. Cambridge University Press
Thomas, S. A. (2000).SSL & TLS Essentials Securing the Web. John Wiley & Sons
Thorsteinson, P. & Ganesh, G.G.A. (2003). .NET Security and Cryptography. Prentice Hall PTR
Tipton, H. F. & Krause, M. (2004). Information Security Management Handbook. CRC Press LLC

Torrubia, A., Mora, F. J. & Marti, L. (2001).Cryptography Regulations for E-commerce and Digital Rights Management. Computers & Security, 20(8), 724-738

Turban, E., King, D., Viehland, D. & Lee, J. (2006). "Electronic Commerce 2006: A Managerial Perspective". Pearson Education Inc., Upper Saddle River, New Jersey

Vaudenay, S. (2006). A Classical Introduction to Cryptography: Applications for Communications Security. Springer Science+Business Media, Inc.

Wagner, D. & Schneier, B. (1996). Analysis of the SSL 3.0 Protocol. *The Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Press

Wang, X., Yin, Y.L. & Yu, H. (2005). Finding Collisions in the Full SHA-1. Advances in Cryptology – Crypto'05

Zacker, C. (2006). MCSE Self-Paced Training Kit (Exam 70-293): Planning and Maintaining a Microsoft Windows Server 2003 Network Infrastructure, Second Edition. Microsoft Press