

Authenticated Trusted Server Controlled Key Establishment

Astha Keshariya

A thesis submitted for degree of
Doctor of Philosophy



At the University of Otago, Dunedin
New Zealand

Date: 31st Dec 2010

This page is left blank.

Keywords

Three Party Authenticated Key Establishment, Trusted-server based Key Establishment Protocols, Authentication, Computer Security, Communications Security, Cryptography, Distributed Denial of Service resilience, Key Agreement Protocols, Key Establishment Protocols, Key Transport Protocols, Asymmetric Key Encryption, RSA based Authentication, Certificate based Authentication, Non-repudiation, Key escrow, Secure Mobile environment, Mobile Payments Systems, Provably Secure Protocols, 3-D Secure Protocol, Securing Mobile Communications, Secure Electronic Transactions, Properties of Key Establishment, Securing Real-time Wireless communications.

Abstract

The trusted server based key establishment protocols are well received by the research community. In this thesis we have discussed the benefits of asymmetric key based authentication scheme mediated by a trusted server which is known to all the users in a system. We have proposed a new trusted server based key establishment protocol (and named it AK-protocol) that makes use of well known certificate based authentication scheme (or ID based scheme when medium level of security is required), and the session key generation requires equal contribution of the trusted server and the participating clients. That is, the generation of ephemeral keys exclusively lies with the trusted server and the generation of a session key is completed only after clients have exchanged their ephemeral keys.

We have analysed the AK-protocol for various properties, e.g., Perfect Forward Secrecy, Known Session-Key Security, Unknown Key Share Resilience, Key Control, Key Freshness, Key Compromise, Bandwidth Required, Scalability, Key Distribution, Central Directory Service, Non-Repudiation, Key Escrow, Desired properties from Three Party Authenticated Key Establishment (3PAKE) protocols and the Message Flow of the AK-protocol. We have also scrutinized the resilience of the AK-protocol when under different attack situations like Replay, Impersonation, DDoS attacks, including a specific situation where an attacker can craft protocol messages to mislead the clients. We have computed its Bit Complexity and evaluated the efforts required to carry out its Cryptanalysis.

We have illustrated its practicability in different arenas. We executed a proof-of-concept implementation of the AK-protocol using Java on TCP, which showed us comparable results with SSL when the trusted server and

the participating clients were in the same network. We substantiated that it can be integrated with the existing 3-D Secure Protocol of Visa and MasterCard for online payment systems which when applied offers more reliable communication, cryptographically. We have also corroborated that the AK-protocol can be implemented with mobile payment systems with worked out examples of cryptographic mathematics involved in the protocol. Additionally, we have also suggested the use of AK-protocol in securing real-time mobile communications where the session key is generated using our protocol and a stream cipher algorithm, RC4 is used for encryption/decryption. We present three examples that illustrate the data flow, cryptographic mathematics involved in the AK-protocol.

Acknowledgments

A special thanks to the University of Otago, for giving me the opportunity to pursue my PhD from their prestigious faculty of learning organization.

I am privileged to have worked under the very knowledgeable supervisor Dr. Hank Wolfe. I want to take this opportunity to express my sincere gratitude to him for his supervision and interest in my thesis. Not only he showed me apt direction but also encouraged me to nourish my thoughts. I am grateful to my co-supervisors Dr. Noria Foukia and Dr. Mariusz Nowostawski for their valuable suggestions and recommendations.

A Special thanks to the Department of Information Science for providing me the resources to accomplish my work. I am grateful to Stephen Hall-Jones, Graham Copson and his entire team from the department, for their endless support.

From the financial perspective, I am indebted to Technology for Industry Fellowships (TIF) and Telecom New Zealand, for funding my research, without which I could not have been able to undertake the research.

My husband Sandip Bose has been a source of inspiration and constant support at every instant, especially when I thought of giving up. This thesis would have not been possible if he was not with me. Special thanks to my entire family who have encouraged me and blessed me in all my endeavours.

I sincerely thank God for showing me the light in his own surreptitious ways.

Table of Contents

| | |
|---|--------|
| <i>Keywords</i> | (iii) |
| <i>Abstract</i> | (iv) |
| <i>Acknowledgements</i> | (vi) |
| <i>List of Figures</i> | (xii) |
| <i>List of Tables</i> | (xiii) |
| <i>Notations</i> | (xiv) |
| <i>Abbreviations</i> | (xvi) |
| | |
| Chapter 1 Introduction..... | 1 |
| 1.1 Chapter Introduction | 1 |
| 1.2 Key Establishment..... | 1 |
| 1.3 Key Distribution Problem | 2 |
| 1.4 Trusted Server based Key Establishment..... | 2 |
| 1.5 Need for Certificates based PKI | 5 |
| 1.6 Research Objectives and Deliverables | 6 |
| 1.7 Structure of the Thesis..... | 8 |
| Chapter 2 Cryptographic Components | 11 |
| 2.1 Chapter Introduction | 11 |
| 2.2 Cryptography | 11 |
| 2.3 Cryptographic Components | 12 |
| 2.3.1 Encryption and Decryption | 12 |
| 2.3.2 Symmetric and Asymmetric Key Cryptography | 12 |
| 2.3.3 Hash Functions..... | 13 |
| 2.3.4 Digital Signatures | 14 |
| 2.4 Establishing Secure Communication..... | 14 |
| 2.5 RSA Algorithm | 15 |

| | |
|---|----|
| 2.5.1 RSA Key Generation | 15 |
| 2.5.2 RSA Encryption and Decryption | 16 |
| 2.5.3 RSA Sign and Verify | 16 |
| 2.6 The integer factorization problem | 18 |
| Chapter 3 Key Establishment | 19 |
| 3.1 Chapter Introduction | 19 |
| 3.2 Key Establishment..... | 19 |
| 3.3 Key Pre-distribution | 20 |
| 3.4 Two-Party Key Establishment | 22 |
| 3.5 Trusted Third Party Key Establishment..... | 23 |
| 3.5.1 Kerberos..... | 25 |
| 3.6 Key Agreement Protocols based on Symmetric-Key Techniques | 28 |
| 3.6.1 ISO/IEC 11770-2 | 28 |
| 3.6.2 Gong's Alternative Protocol..... | 30 |
| 3.7 Key Transport Protocols based on Public Key Encryption | 30 |
| 3.7.1 Needham-Schroeder Public-key protocol..... | 31 |
| 3.7.2 X.509 Strong Authentication Protocol | 31 |
| 3.7.3 SSL/TLS protocol | 32 |
| 3.8 Key Transport Protocols..... | 34 |
| 3.8.1 Beller-Yacobi Protocol..... | 34 |
| 3.9 Key Agreement Protocols based on Asymmetric Key Techniques | 35 |
| 3.9.1 Diffie-Hellman Key Agreement | 35 |
| 3.10 Authentication in Three Party Key Agreement Protocols..... | 36 |
| 3.11 Properties of Key Establishment Protocols..... | 37 |
| Chapter 4 Public Key Infrastructure..... | 41 |
| 4.1 Chapter Introduction | 41 |
| 4.2 Public Key Infrastructure | 41 |
| 4.3 Certification & Registration Authority, and Certificate Directory | 42 |
| 4.4 Digital Certificate | 43 |

| | |
|---|----|
| 4.5 PKI Functions..... | 44 |
| 4.5.1 Certificate Binding or Certificate Issuance..... | 44 |
| 4.5.2 Certificate Verification | 45 |
| 4.5.3 Certificate Revocation..... | 45 |
| 4.6 Hierarchical and Non-Hierarchical Certification | 46 |
| 4.7 Certificate Chain..... | 46 |
| 4.8 Cross Certification | 47 |
| 4.9 X.509 Certificates | 47 |
| 4.9.1 Contents of X.509 Certificates | 47 |
| Chapter 5 AK-Protocol Description | 50 |
| 5.1 Chapter Introduction | 50 |
| 5.2 The Basis of AK-protocol | 51 |
| 5.3 The AK-protocol Overview | 52 |
| 5.3.1 System Setup and Key Generation | 55 |
| 5.4 Roles of TS, CA and RA | 57 |
| 5.5 The Process of Authentication and Key Establishment | 59 |
| 5.5.1 Distributed Environment with Multiple TTPs | 64 |
| 5.6 Mathematical Computations | 65 |
| 5.7 Deployment in a Closed Environment | 67 |
| 5.8 Properties of the Protocol | 69 |
| 5.8.1 Perfect Forward Secrecy | 70 |
| 5.8.2 Known Session-key Security | 71 |
| 5.8.3 Unknown Key-Share Resilience | 71 |
| 5.8.4 Key-Control | 72 |
| 5.8.5 Desired properties in a 3PAKE | 72 |
| 5.8.6 Key freshness and Key compromise | 74 |
| 5.8.7 Bandwidth required | 75 |
| 5.8.8 Scalability | 75 |
| 5.8.9 Key distribution..... | 76 |

| | |
|--|-----|
| 5.8.10 Centralized Directory Service..... | 76 |
| 5.8.11 Non-Repudiation..... | 77 |
| 5.8.12 Message Flow | 78 |
| 5.8.13 Key Escrow | 79 |
| 5.9 Security Analysis of the protocol | 79 |
| 5.9.1 Replay Attack | 79 |
| 5.9.2 Bit Complexity | 81 |
| 5.9.3 Impersonation Attack | 83 |
| 5.9.4 When an Adversary can Craft Protocol Messages..... | 83 |
| 5.9.5 Distributed Denial of Service (DDoS) Attacks | 85 |
| 5.9.6 Cryptanalysis | 90 |
| 5.9.7 Integer Factorization and RSA Problem | 91 |
| Chapter 6 Implementations | 93 |
| 6.1 Chapter Introduction | 93 |
| 6.2 Implementation of the AK-protocol | 94 |
| 6.3 Integrating the AK-protocol with 3Dsecure™ Model..... | 101 |
| 6.3.1 3-D Secure Model | 102 |
| 6.3.2 3-D Secure Protocol Messages | 103 |
| 6.3.3 Process of Payment in 3-D Secure | 104 |
| 6.3.4 Integrated with the AK-protocol..... | 105 |
| 6.4 AK-protocol in Wireless Environment..... | 112 |
| 6.4.1 Standard Mobile Payment System..... | 112 |
| 6.4.2 Integration of Mobile Payment System and the AK-protocol | 114 |
| 6.4.3 Real-Time secured communication with AK-protocol..... | 116 |
| 6.5 Examples of the AK-protocol | 119 |
| 6.5.1 A Simple Example..... | 119 |
| 6.5.2 Example RSA-123 Bits | 121 |
| 6.5.3 Example with RSA-140 Bits | 123 |
| Chapter 7 Conclusions and Further-Work..... | 126 |

| | |
|---------------------------------|-----|
| 7.1 Summary of the Thesis | 126 |
| 7.2 Further Work | 132 |
| References | 134 |
| Appendix A | 157 |
| Appendix B..... | 162 |

List of Figures

| | |
|--|-----|
| Figure 1: TLS Hand shake..... | 33 |
| Figure 2: Sample Self-Signed Certificate..... | 49 |
| Figure 3: The Protocol Overview | 55 |
| Figure 4: Distributed Environment with Multiple TTPs | 64 |
| Figure 5 : The Protocol in Mobile Environment | 69 |
| Figure 6: Adversary capturing the protocol messages. | 80 |
| Figure 7 : Signed Protocol messages with two-way authentication..... | 84 |
| Figure 8: DB of Clients' information | 94 |
| Figure 9: Message Flow of the Implementation on TCP | 95 |
| Figure 10: DB of the transactions..... | 96 |
| Figure 11: Snapshot of Server Module..... | 97 |
| Figure 12: Snapshot of Client A..... | 98 |
| Figure 13: Snapshot of Client B | 99 |
| Figure 14: Protocol Analysis | 100 |
| Figure 15: Protocol Analysis with increasing threads..... | 101 |
| Figure 16: 3D Domain of Visa | 103 |
| Figure 17: 3D Secure integrated with the protocol | 108 |
| Figure 18: Standard Mobile Payment System | 113 |
| Figure 19: Encrypting Real-time Communication with RC4..... | 118 |
| Figure 20: Sample 2 Snapshot of Server Module..... | 162 |
| Figure 21: Sample 2 Snapshot of Client A..... | 163 |
| Figure 22: Sample 2 Snapshot of Client B | 163 |

List of Tables

| | |
|---|-----|
| Table 1: Euclidean algorithm for calculating gcd of two numbers | 158 |
| Table 2: RSA Key Generation..... | 15 |
| Table 3: RSA Encryption/Decryption | 16 |
| Table 4: RSA Sign/Verify | 17 |
| Table 5: RSA Signing the Hash and its Verification..... | 17 |
| Table 6: ISO/IEC 11770-2 Key Establishment Mechanism 12 | 29 |
| Table 7: ISO/IEC 11770-2 Key Establishment Mechanism 13 | 29 |
| Table 8: Gong's Alternative Protocol | 30 |
| Table 9: Needham-Schroeder public-key protocol | 31 |
| Table 10: X.509 Two-way authentication protocol..... | 32 |
| Table 11: Beller-Yacobi 4-pass key transport protocol..... | 34 |
| Table 12: Basic Diffie-Hellman Key Agreement..... | 35 |
| Table 13 : Mathematical computations required at server/clients end | 66 |
| Table 14 : Types of Protocol Messages..... | 78 |
| Table 15: 3D Secure Protocol Messages | 104 |
| Table 16: Protocol Messages in our Scheme..... | 106 |

Notations

| | |
|-----------------------------|---|
| η | $\eta = P.Q$, Modulo for RSA encryption/decryption. |
| (d_A, η) | Mathematical representation of Private Key of Client A |
| (e_A, η) | Mathematical representation of Public Key of Client A |
| $(\text{message})_{K_{XY}}$ | Message encrypted using symmetric encryption where K_{XY} is the shared key between entity X and Y. |
| , | Concatenation. |
| . | Multiplication. |
| [message] | An optional message. |
| $\{\text{Text}\} P_A$ | Text is encrypted with Public key of Client A. |
| $\{x\} S_{TS}$ | Integer x Signed by Trusted Server. |
| CertA | Public Key Certificate of Client A. |
| Client A | The user A subscribed in the system. |
| $H(x)$ | Hash of integer x. |
| K_{XY} | The key intended to be shared by entity X and Y. |
| m | Generator of Z_N^* Cyclic Group over modulo N. |
| m' | Generator of Z_η^* Cyclic Group over modulo η . |
| Message X | Encrypted key-parts to Client X |
| N, p | Prime numbers used for Modulo operation. |
| n_x | Nonce generated by entity X. |
| N_X, N'_X | A sequence number issued by entity X |
| P_A | Public Key of Client A. |
| Pvt_A | Private key of the Client A |
| Req B | Request to initialize a session with Client B. |

| | |
|--------------------------|--|
| Req CertB | Request Certificate of Client B. |
| R_X, R'_X | A random number chosen by entity X. |
| r_X, r_Y | Real numbers r_X and r_Y that are never re-used. |
| TS | Trusted Server. |
| TV P_X | A time variant parameter, such as a random number, a timestamp or a sequence number generated by entity X. |
| T_X, T'_X | A timestamp issued by entity X. |
| T_X/N_X | A timestamp or a sequence number issued by entity X. |
| $X \rightarrow Y$: Data | Data is send from entity X to Y. |
| ϕ | Phi denoted by $\phi = (P-1).(Q-1)$ for RSA Key Generation. |
| $A \gg B$: Data | Data is sent from entity A to B via some secondary channel, e.g., courier, phone, etc. |
| x^{-1} | Multiplicative Inverse of an integer x. |

Abbreviations and Acronyms

| | |
|-------------|---|
| 2P | Two Party |
| 2PAKE | Two party Authenticated Key Exchange |
| 3-D Secure™ | Three Domain-Secure protocol by Visa and MasterCard |
| 3P | Three Party |
| 3PAKE | Three party Authenticated Key Exchange |
| 3PEKE | Three party Encrypted Key Exchange |
| 3PKD | Three party key distribution |
| AES | Advanced Encryption Standard |
| AKA | Authenticated Key Agreement |
| AKE | Authenticated Key Establishment |
| API | Application Programming Interface |
| CA | Certificate Authority |
| CRL | Certificate Revocation List |
| DES | Data Encryption Standard |
| DER | Distinguished Encoding Rules |
| DH | Diffie-Hellman key agreement protocol |
| DLP | Discrete Logarithm Problem |
| ECC | Elliptic Curve Cryptography |
| EKE | Encrypted Key Exchange |
| GSM | Global System for Mobile Communications |
| HMAC | Keyed Hash Message Authentication Code |
| HTTP | Hypertext Transfer protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ID | Identifier |

| | |
|----------|---|
| ID-Based | Identity based |
| IETF | Internet Engineering Task Force |
| KDC | Key Distribution Centre |
| KE | Key Establishment |
| KGC | Key Generation Centre |
| LDAP | Lightweight Directory Access Protocol |
| MAC | Message Authentication Code |
| MD | Message Digest |
| MD5 | Message Digest algorithm 5 |
| PKE | Public key Encryption |
| PKI | Public Key Infrastructure |
| PKIX | Public Key Infrastructure based on X.509 Certificates |
| RA | Registration Authority |
| RFC | Request for Comments |
| RSA | Public key algorithm by Rivest Shamir Adleman |
| SHA | Secure Hash Algorithm |
| SSL | Secure Socket Layer |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| TS | Trusted Server |
| TTP | Trusted Third Party |
| UDP | User Datagram Protocol |
| UID | User Identification |
| UML | Unified Modelling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |

Chapter 1

Introduction

1.1 Chapter Introduction

This chapter introduces the key establishment protocols and key distribution problem while focusing on certificate based authentication and trusted server based key establishment. We briefly convey research deliverables and the structure of the thesis.

1.2 Key Establishment

Key establishment (KE) is a cryptographic mechanism whereby a secret key becomes available to two or more parties. For two parties to communicate securely over an insecure channel, they must be able to authenticate one another and establish a common session key.

Key Establishment Protocols are typically used at the beginning of communication between parties, to establish a mutually agreed *session key* which enables a secured communications further, during the entire session.

Key establishment can be further divided into two categories: *Key agreement protocols*, where the session key is derived by two (or more)

parties as a function of information contributed by, or associated with each of these such that neither party can predetermine the resulting value. *Key transport protocols*, where one party generates the session key and then securely transfers it to the other party.

1.3 Key Distribution Problem

For peer-to-peer secure communication in a distributed network, a simple solution is to let every entity share a distinct long term key with each of its partners. However, it turns out that this solution has poor scalability and incurs high cost of secret long term key maintenance, because in a distributed network, at any instance there can be random amount of new entities joining in and old entities dropping out.

Consequently, it is a difficult problem to establish a common key between two clients without having them share any prior secret-key. This problem was solved with Diffie-Hellman key agreement scheme [Diffie-Hellman|76]. Though, the scheme is exposed to the possibility of man-in-the middle attack [Schneier|95]. Later, different approaches and protocols have been developed to solve the problem [Dutta|05], [Menezes|96].

1.4 Trusted Server based Key Establishment

In *three party authenticated key exchange* (3PAKE) protocols, each client shares a secret only with a trusted server who assists in generating a session key used for securely sending messages between two participating clients. Compared with *two-party authenticated key exchange* (2PAKE) protocols where each pair of parties must share a secret with each other, a three-party protocol offers better solutions to the key management problem. The

3PAKE protocol can be employed within various applications, e.g., a trusted server assists in transactions between buyer and seller in e-commerce, home location register (HLR) helps caller's dial with visited location register (VLR) over telecommunication, etc., for mutual authentication and secure communication.

Recently, Chen et al. [Chen-Lee|08] presented an analysis of 3PAKE protocols and pointed out their deficiencies, which they suggested their protocols will withstand. They also concluded that there are three issues in 3PAKE protocols which can be further improved: (1) to reduce latency, communication steps in the protocol should be as parallel as possible, with fewer rounds required to establish a session key, the protocols can reduce communication latency and achieve a quick response; (2) existence of a security-sensitive information at the server side may lead adversary to attack the server to retrieve critical keying material, which should be removed from the system; (3) resources required for computation should be as few as possible to avoid the protocol becoming an efficiency bottleneck.

Key Distribution Centre (KDC) based protocols includes a resource-rich trusted third-party for the process of key establishment, for example, SPINS [Szewczyk|01], KryptoKnight [Molva|92] and Kerberos [Schiller|88] follow the *three-party encrypted key exchange* (3PEKE) approach, where each registered client keeps only one secret with a trusted server. Any pair of registered clients can establish authenticated session keys with the help of the server. These protocols require lesser memory resources since each party only needs to secure its communications with the KDC making use of a single symmetric (or asymmetric) key shared between them before establishing a secured communication.

However, recently 3PEKE protocols were analyzed in [Chien|09], [Lo-Yeh|09] and as a matter of fact, the advent of various attacks like password guessing attacks (both online and offline), replay attacks, man-in-the-middle attacks, etc., on *weak* passwords based systems has put all “sharing passwords with the trusted server” based schemes at risk. This has brought substantial weight to the use of asymmetric keys, which are shared between trusted server and clients.

These pre-shared (symmetric) key based schemes would require the clients to know each other on a *level of trust* to have a key shared between them. While trusted third-party based systems shifts the element of trust to a well-known *Trusted Server*.

The authentication in a key agreement protocol can be divided into two broad categories: certificated-based [Riyami|02, Riyami|03, Cheng|04, Joux|03, Shim|03, Aivaloglou|08 and Buttyán|10] and ID-based [Chien|04, Liu-Zhang|03, Nalla|03, Nalla|03, Shim|03b, Zhang-Liu-Kim|02, Tso|05, Boneh|01, Smart|02, McCullagh|05, Wang|05, Cheng|05, Sakai|00 and Gorantla|08].

Cheng et al. [Cheng|04] proposed *two* tripartite key agreement protocols: one is certificate-based and the other is ID-based. They also analyzed their schemes against a list of attacks. However, some of their attack assumptions are not practical, but their schemes cannot resist the practical insider impersonation attack and their ID-based version even discloses the entities’ private keys. Juang [Juang|04] proposed a three-party key exchange scheme which essentially uses smart cards which are authenticated by the trusted server to further generate a nonce-based session key. Later, many of tripartite identity based key establishment protocols were studied in [Hölbl|09, Wanga|09 and Guo|09] and concluded that Identity-based

authenticated key agreement protocols can be an alternative for certificate-based protocols when efficient key management and moderate security is required.

1.5 Need for Certificates based PKI

Any business transaction requires some level of trust between its participants. Such a trust relationship might be established by a variety of means including past experience or exchange of documents or credentials that can assist them to establish a secure communication, later [Ion|10].

Public Key Infrastructure (PKI) has been used in a wide variety of distributed applications ranging from e-commerce and web services applications to complex systems such as Grid computing and virtual organizations. PKI has also been used in the design of various security protocols as mode for authentication, such as Secure Socket Layer/Transport Layer Security (SSL/TLS) [Thomas|00], Internet Key Exchange (IKE) [Cheng|01] and Secure Electronic Transaction (SET) [Giampaolo|03], 3-D Secure™ [Carbonell|09], iKP [Bellare|09].

Certificate based PKI has been the source of many of the radical advances in the evolution of security solutions to provide: authentication, authorization, confidentiality, integrity and accountability. The basic goals of certificate based PKI is to provide authentication and confidentiality in distributed systems. Certificates are a reusable component for strong authentication, which offers itself for the solution to scalability and mobility, particularly for large organizations [Haidar|09].

1.6 Research Objectives and Deliverables

In this thesis, we are proposing a new certificate-based key establishment protocol mediated by a Trusted Server, which can be deployed in a distributed system where high level of security is required, such as financial transactions. However, the protocol is open to accommodate any ID-based authentication scheme instead of certificate-based scheme, where medium level of security is required.

We are utilizing the efficient trusted server which functions as an authentication server and ephemeral key generator. The overall architecture of the system offers us layers of abstraction: *the first layer* is the authentication; and *the second layer* is the key establishment. In our protocol, certificates are used by clients as the mode of authenticating themselves. The participating clients are authenticated by the trusted server, after which it generates two distinct but related ephemeral keys and encrypts them with the clients' respective public keys before distributing them. The clients then exchange the ephemeral keys they possess to establish a session key at both the ends.

Once the trusted server has distributed the ephemeral keys to the participating clients they are removed from the buffers of the system. But, these ephemeral keys may be saved in a secure database accessible only to the trusted server when the system desires to have the capability of non-repudiation and key escrow. For example, it is less probable that a mobile system would want to save the session keys of all the past secure voice communications for the sake of non-repudiation.

The delivery of the ephemeral keys is done securely using asymmetric encryption (RSA). The clients are then required to exchange the ephemeral

key they possess, to finally generate a common session key at both ends. This session key can be used with any symmetric (block or stream cipher) or asymmetric key algorithms to secure further communications.

There can be an argument whether our protocol is a key transport protocol or key agreement protocol, because the server generates two key-parts, and delivers them to the clients securely. However, the clients still have to compute the session key out of these two key parts, so we have placed AK-protocol in the category of key agreement protocols. Additionally, in a key transport protocol one of the participating clients either generates whole or a part of the ephemeral keys, whereas in our protocol the server generates the ephemeral keys and no participating clients can derive the session key out of the single ephemeral key they possess.

With this AK-protocol we are putting forward the concept of three-party key establishment protocol which is controlled by the trusted server not only for authentication but also for ephemeral key generation. In this thesis we have described the concept and functioning of the AK-protocol for different environments such as distributed, closed and trusted. Furthermore, we have also discussed the fundamental security properties that are exhibited by our protocol (see Section 5.8 for more information), e.g., perfect forward secrecy, bandwidth required, etc., along with different attack scenarios (see Section 5.9 for more information) like replay, impersonation, an adversary capable of crafting the protocol messages, distributed denial of service attack and the cryptanalysis of the protocol.

Later, we have also discussed the implementation aspects of the AK-protocol in real life scenarios. (a) We present the Java implementation of our protocol which delivers comparable time as that of SSL to establish a session key between the two entities when the trusted server and the

participating clients were in the same network (see Section 6.2 for more information). (b) We have analysed the existing 3DSecure™ protocol of Visa and MasterCard, and have attempted to integrate our protocol with the existing system keeping in mind, that since the protocol is used at large minimum modifications are suggested that could be of advantage to the existing system (see Section 6.3 for more information). (c) A scheme for mobile payment system, when our protocol is amalgamated with the general payment systems is suggested, (see Section 6.4.2 for more information). Another scheme is presented to secure real-time data, e.g., voice communication where session key is established between two entities using our protocol after which RC4 is used to encrypt/decrypt the real time communication (see Section 6.4.3 for more information).

1.7 Structure of the Thesis

This section briefly outlines the structure of the thesis and presents the overview of individual chapters.

Chapter 2 consists of the basics of various cryptographic techniques and their components, building blocks to secure communication and RSA algorithm for encryption/decryption (Sign/Verify), Integer factorization problem and Discrete Logarithm Problem (DLP), which are referenced at several places in the thesis.

Chapter 3 covers key establishment and need for certificate based system. We focus our attention on three-party based key establishment, however we have incorporated various other forms of key establishment such as symmetric key, asymmetric key and hybrid authentication based on two-party or three-party key establishment systems to have assessment based understanding of what other forms of key establishment protocols offer. We

then discuss the basic concept of the AK-protocol and general expected properties in any key establishment protocol.

Chapter 4 assembles all the relevant details required to understand the Public Key Infrastructure such as, the functioning of Certification Authority (CA), Registration Authority (RA) and Certificate directory services. Basic functions: Certificate issuance, Certificate verification and Certificate revocation, various modes of certification and X.509 certificates.

Chapter 5, in this chapter we illustrate the AK-protocol for distributed and closed environment, its system setup requirements, key generation and mathematical computation required at server and client end. We also discuss the relevant properties exhibited by the protocol, e.g., Perfect Forward Secrecy, Known Session Key security, Unknown Key Share resilience, Key control, Desired properties in any 3PAKE protocols, Required bandwidth, Non-repudiation, Scalability, Key escrow, Key freshness, Key compromise resilience, and Centralized key distribution and directory services. Analysis of the AK-protocol like bit Complexity, Cryptanalysis of the protocol, the protocol message flow, including the system under different attack situations such as Replay, Impersonation, DDoS attacks and a specific situation where an attacker can craft the protocol messages to mislead the clients.

Chapter 6 is comprised of the implementation aspects of the AK-protocol in various practical situations. It is divided into four major sections: First, we discuss the Java implementation of our protocol on TCP layer. Second, we discuss the integration of our protocol with the existing 3-D secure protocol of MasterCard and Visa. Third, we suggest the integration of our protocol in mobile environments: (a) for mobile payment systems and (b) for secure real-time communication by generating the session key using our

protocol and then using a fast stream cipher algorithm like RC4 for encryption/decryption. Fourth, we present three examples illustrating the data flow and the cryptographic mathematics involved in the AK-protocol.

Chapter 7 is the final chapter of the thesis where we present a summary of the thesis and further research work.

Chapter 2 Cryptographic Components

2.1 Chapter Introduction

In this chapter we cover the basics of various cryptographic techniques, their components, building blocks to secure communication, key-establishment, need for a certificate based system, RSA algorithm for encryption/decryption and Integer Factorization problem, required to understand the basics of cryptographic operations.

2.2 Cryptography

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, data origin authentication and non-repudiation.

These cryptographic goals: *Data confidentiality*, ensuring that data is kept secret from all except those who are authorized to know it. *Data integrity* implies ensuring that data is not tampered with by unauthorized means. *Data origin authentication*, proving the validity of the source of data. *Entity authentication*, proving the validity of entity's identity and *Non-repudiation*,

preventing a party from denying previous actions or commitments, forms the basis of cryptographic science.

2.3 Cryptographic Components

By making use of the cryptographic techniques such as encryption, decryption, digital signature, hash, etc. we can achieve cryptographic goals such as, confidentiality, integrity, authentication and non-repudiation.

2.3.1 Encryption and Decryption

In cryptography, *encryption* is the process of transforming information into encrypted text (cipher-text) which is unreadable to anyone except those possessing special knowledge, usually referred to as a key, by making use of algorithms like AES, DES, RSA, ECC. The reverse process, *decryption* converts encrypted information to readable form or plaintext. Encrypting data transmitted over an insecure channel ensures the confidentiality of the message. However, other techniques are still needed to protect the integrity and authenticity of a message, for example, verifying the MAC (message authentication code) ensures integrity of the message and a digital signature ensures the authenticity of the message.

2.3.2 Symmetric and Asymmetric Key Cryptography

Symmetric-key cryptography involves symmetric algorithms, which use either related or identical cryptographic keys for both encryption and decryption. The key is a shared secret between two or more parties that can be used to secure further communications between them. To use a

symmetric encryption scheme, the sender and receiver must securely share a key in advance. Other terms for symmetric-key encryption are secret-key, single-key, shared-key, one-key, and private-key encryption. In practice, symmetric-key algorithms are generally much less computationally intensive than asymmetric key algorithms.

Public-key cryptography involves the use of asymmetric key algorithms. Unlike symmetric key algorithms, they do not require a secure initial exchange of one or more secret keys between both sender and receiver. The asymmetric key algorithms are used to create a mathematically related key pair: a secret private key and a published public key. The sender encrypts the message with the public key of the receiver and at the other end when receiver gets the cipher text he decrypts it with his private key.

Confidentiality can be achieved by encrypting the message using the public key, which can only be decrypted using the private key. Use of these key pairs allows more than encryption and decryption, for example, authenticity of a message by creating a digital signature of a message which can be verified by anyone who possesses sender's public key.

2.3.3 Hash Functions

A cryptographic *hash function* (h) is a one-way function, that is, given a value $y \in \{0, 1\}^n$ where n is the bit length, it is computationally infeasible to find an m with $h(m) = y$. The hash function must also have a property, given a message m , it is computationally infeasible to obtain a second message m' with $m \neq m'$ and $h(m) = h(m')$. This property is called the second pre-image resistance. Such a pair of messages (m, m') , with $m \neq m'$ and $h(m) = h(m')$ is called a collision of h and a cryptographic hash function should be

collision resistant. Some popular hash functions are MD5 [RFC 1321], SHA-224 [RFC 3874] and RIPEMD-160 [RFC 2857].

A very important application of hash functions is to ensure the integrity of the message and data origin authentication. When a hash function is used in conjunction with digital signature schemes, i.e., the message is first hashed, the generated value is called *message authentication code* (MAC), and this MAC is signed by the private key of the sender. This signed MAC is sent along with the original message to the receiving end which ensures the integrity of the message, and confirms that the message is actually created by the sender, i.e., *message authentication or data origin authentication*.

2.3.4 Digital Signatures

When a message is signed with a sender's private key it can be verified by anyone who has access to the sender's public key, thereby proving that the sender had access to the private key (and therefore is likely to be the person associated with the public key used), and that the message is actually created by sender [Rivest-Shamir-Adleman|78].

2.4 Establishing Secure Communication

A fundamental problem in cryptography is how to communicate securely over an insecure channel, which might be controlled by an adversary. Two well understood two phases involved in the process of establishing secure channel in a distributed system: (i) *Initialization phase*, where a secondary trusted communication channels (like, trusted couriers, personal registration at a trusted centre, mutual authentication by speaker identification on a voice channel, etc.) are used for setting up security parameters. These security parameters include symmetric or asymmetric keys, algorithms for

encryption/decryption, hash algorithm, etc. (ii) *Communication phase*, where entities (users or applications) can typically communicate over insecure channels using security parameters decided in the previous phase.

2.5 RSA Algorithm

One of the most popular public key algorithms, RSA is named after its inventors R. Rivest, A. Shamir and L. Adleman [Rivest-Shamir-Adleman|78], and provides both confidentiality and data origin authentication. Its security is based on the intractability of integer factorization, discussed in Section 2.6.

2.5.1 RSA Key Generation

RSA public key cryptosystem makes use of pair of keys called public key and private key. *Public key* which is known to all, is used for encryption and *private key* which is known only to the owner, is used for decryption. The generation of these two RSA public and private keys is shown in the table below.

Table 1: RSA Key Generation

- | |
|---|
| <ol style="list-style-type: none"> 1. Generate two large prime numbers P and Q. 2. Compute $\eta = P.Q$ and $\phi = (P-1).(Q-1)$. 3. Select a random number e such that $\gcd(e, \phi) = 1$. 4. Compute d such that $e.d \equiv 1 \pmod{\phi}$. 5. Public key (e, η) and Private key (d, η). |
|---|

2.5.2 RSA Encryption and Decryption

After these key pairs (public and private keys) are generated they can be used for the process of encryption and decryption, as shown in Table 3. Let's say user A's public key (e, η) is available to all, so anyone who wants to send a secured message to the user A can encrypt his message m with A's public key and send it to A, which upon receiving, A can decrypt using his private key (d, η) [PKCS1].

Table 2: RSA Encryption/Decryption

| | |
|-------------------|--|
| Encryption | <ol style="list-style-type: none"> 1. Acquire A's authentic Public key (e, η) 2. Compute $C = m^e \bmod \eta$ where m is the message and C is the cipher text and send it to A. |
| Decryption | <ol style="list-style-type: none"> 1. A computes $m = C^d \bmod \eta$ where Private key is (d, η) upon receiving the cipher text. |

2.5.3 RSA Sign and Verify

As shown in Table 4, the same pair of keys can be used for signing the message and then verifying it. Signing a message ensures integrity and data origin authentication of the message. However, in a real situation the entire message is not signed but a hash of the message is signed. On the receiving end after decrypting the message, its hash is calculated and verified against the signed hash appended with it, this gives surety that the message was actually created by the sender and was not tampered with in between. This mechanism serves for integrity and data origin authentication.

To compute the signature of the message m , it is encrypted by the private key of the sender which can be verified on the other end by decrypting it using sender's public key.

Table 3: RSA Sign/Verify

| | |
|---------------|--|
| Sign | 1. Sender A computes $S = m^d \bmod \eta$ where m is the message and S is the signature generated, which is sent along with the message. |
| Verify | 1. Receiver acquires A's authentic Public key (e, η) . 2. Compute $m = S^e \bmod \eta$. |

In a case where the hash of the message is signed to ensure the integrity of the message, the generated value is called *Message Authentication Code (MAC)*. The generated MAC is sent along with the message securely so that the receiving party can verify it with the MAC generated at its end, as shown in the Table 5.

Table 4: RSA Signing the Hash and its Verification

| | |
|---------------|---|
| Sign | 1. Sender A computes hash of the message $H(m)$ then signs the hash $S_H = (H(m))^d \bmod \eta$ where S_H is the generated MAC, which is sent along with the message. |
| Verify | 1. Receiver acquires A's authentic Public key (e, η) . 2. Computes hash of the message received $H(m')$, and 3. Encrypts the received MAC, $H(m) = (S_H)^e \bmod \eta$. When $H(m') = H(m)$ then the message is said to be verified. |

2.6 The integer factorization problem

Definition 1 The *integer factorization problem or factoring*, is finding prime factors of a given positive integer n , where n can be denoted as $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_{k-1}^{e_{k-1}} \cdot p_k^{e_k}$, where p_i are distinct primes and e_i are positive integers. The problem of deciding whether an integer is composite or prime is much easier than the factoring problem [Menezes96]. Hence, an integer should be tested for its *primality* before attempting to factor an integer.

Definition 2 The *RSA problem*: given a positive integer $\eta = P \cdot Q$ where P and Q are odd prime numbers, is finding an integer m such that $m^e = c \pmod{\eta}$ where $\gcd(e, (P-1)(Q-1)) = 1$ and c is some integer.

Definition 3 The *Discrete Logarithms Problem (DLP)*: let G be a cyclic group of prime order p with a generator g and x is randomly chosen in Z_p . Consider the equation $y = g^x \pmod{p}$. Given g , x and p , it is a straightforward equation to calculate y . However, given y , g and p , it is computationally infeasible to calculate x [Menezes96].

Chapter 3 Key Establishment

3.1 Chapter Introduction

In this chapter, we focus our attention on three-party based key establishment, however we have incorporated various other forms of key establishment such as two-party and three-party based, symmetric key, asymmetric key and hybrid authentication and key establishment systems, to have assessment based understanding of what other forms of key establishment protocols offer. We then discuss the basic concept of our protocol.

3.2 Key Establishment

Key establishment protocols enable two or more parties to authenticate each other and establish cryptographic keying material between them, through a public network, which subsequently can be used for cryptographic endeavours. Key establishment includes two subcategories, *key transport* where one party creates or otherwise obtains a secret value and securely transfers it to the others; and *key agreement*, in which a shared secret is derived by two (or more) parties as a function of information contributed by,

or associated with, each of these, such that no participating party can predetermine the resulting value [Menezes|96].

The cryptographic keys derived from the key establishment protocol are then typically used to secure the communication channel by providing confidentiality and integrity services. Achieving any form of authentication in key exchange protocols inevitably requires some secret information (like cryptographic keys) to be established between the communicating parties prior to the authentication phase. Therefore, designing and analyzing key agreement protocols is a non-trivial task.

In 1976, Diffie and Hellman proposed the first key agreement protocol [Diffie-Hellman|76]. However, the basic Diffie-Hellman protocol does not authenticate the two communicating entities, thus is susceptible to the man-in-the-middle attack. Later, different approaches and protocols have been developed to solve the problem [Dutta|05], [Menezes|96].

3.3 Key Pre-distribution

Key pre-distribution schemes are key establishment protocols whereby the resulting established keys are completely determined a priori by initial keying material, for example, one-time pad. In contrast, *dynamic key establishment or session key establishment* schemes are those whereby the key established by a group of users varies on subsequent executions. In this case the session keys are dynamic and it is usually intended that the protocols are immune to known-key attacks.

Group key establishment protocols allow a number of individuals to securely exchange cryptographic keys or establish a shared key among them using an insecure medium [Studer|08]. However, many key establishment

protocols involve a centralized or trusted party, for either (or both) initial system setup and on-line actions, i.e., involving real-time participation, between two parties. This party is referred to by a variety of names depending on the role played, like *trusted third party (TTP)*, *trusted server (TS)*, *authentication server (AS)*, *key distribution centre (KDC)* and *key translation centre (KTC)*. If the protocol can ensure to the participating clients that they are sharing the key with the identified clients, then the key agreement protocol is called *authenticated*¹. Moreover, authenticated key agreement protocols offer *implicit authentication*. Additionally, *explicit key authentication* implies that an identified party actually possesses the specified key.

When implicit key authentication is provided to both participating parties in a key establishment protocol, the protocol is said to provide *mutual key authentication*. The phrase *unilateral key authentication* is used when one specifically speaks of authentication in only one direction.

3.3.1 Cryptographic Keys

Cryptographic keys can be either secret keys for *symmetric cryptography* [Jun|06] or a pair of keys (private and public) for *asymmetric cryptography* [Katz|05], which may be one of the forms of underlying secret information, pre-established between the parties.

Cryptographic keys are expected to be random, or unpredictable which becomes hard for humans to remember entailing a significant amount of administrative work and cost. Eventually, it is because this drawback that password-based authentication came into existence. Although, passwords

¹ Key authentication is independent of the actual possession of the key by the parties, for this reason, it is sometimes referred to more precisely as implicit key authentication.

are drawn from a relatively small space like a dictionary, and are easier for humans to remember than cryptographic keys.

3.4 Two-Party Key Establishment

Bellovin and Merritt [Bellovin|92], were first to consider password-based authentication over a public network, and agree on a cryptographic key to be used for protecting their subsequent communication. Their protocol, known as encrypted key exchange (EKE), was a great success in showing how one can exchange password authenticated information while protecting poorly-chosen passwords from the notorious off-line dictionary attacks. This initial work has been followed by a number of two-party protocols [Boyko|00], [Bellare|00], [Katz|01], [IEEEa], [Zhang|04], [Strangio|06] and [Jiang|05] offering various levels of security and complexity.

The Simple Authenticated Key Agreement (SAKA) protocol proposed by Her-Tyan Yeh et al. [Tyan|02] is also a two party key agreement protocol based on password based authentication and Diffie-Hellman key agreement. However, Tseng [Tseng|00] discovered a problem in the SAKA protocol that an unauthorized person could use the session key and deceive the user. He proposed a solution to the problem correcting the key verification phase. Ku-Wang [Ku|00] proved that this solution could also be attacked in two ways, and tried to overcome these attacks by changing the message used in the verification phase. Seo-Sweeney's protocol [Seo-Sweeney|99] also had weaknesses in session key verification phase as indicated in [Tseng|00] [Lin|00]. Later, Kim et al. [Kim|04] analysed all these protocols and proposed their protocol with improved performance by combining the separated phases for safe authentication.

While password based *Two-Party Authenticated Key Exchange* (2PAKE), are well suited for client-server architectures, they tend to become inconvenient and costly when implemented in large scale systems, since 2PAKE protocols require each pair of potential communicating parties to share a password. Due to this problem, three-party models, where each user or client share a single password with the trusted server, have been often used in designing PAKE protocols [Gong-Lomas|93], [Steiner-Tsudik|95], [Kwon|99], [Lin-Sun|00], [Abdalla|05] and [Kim|09]. Typically, trusted server assists clients in establishing a session key by providing authentication services to them.

3.5 Trusted Third Party Key Establishment

In literature, numerous server-based key establishment protocols exist. A number of them are based on the idea of the Needham and Schroeder Protocol [Needham|78]. The most important and a de-facto standard, is the Kerberos protocol [Kohl|93]. All server based protocols have in common that the peers share a secret key with a Trusted Third Party (TTP), which is responsible to negotiate a session key between the peers. In some protocols the server is the single instance that controls the input for session key generation, in other protocols either one peer, both peers or the TTP and both peers deliver input to the key derivation function.

According to [Menezes|96] and [Boyd|03] the following protocols achieve similar or stronger security in comparison to the Kerberos: Yahalom Protocol [Burrows|96], 3PKDP and Optimized 3PKDP [Janson|93], Gong Alternative Protocol [Gong|93], Boyd Four-Pass Protocol [Boyd|96].

In 1995, Steiner, Tsudik and Waidner [Steiner-Tsudik|95] proposed a *three-party Encrypted Key Exchange* (3PEKE) protocol also known as STW-

3PEKE in which all the clients share a password with a trusted server which mediates between two communicating parties to allow their mutual authentication. In [Lin-Sun|00], Lin et al. pointed out that the STW-3PEKE is not only vulnerable to undetectable online password guessing attacks but also vulnerable to an offline password guessing attack. They also proposed a 3PEKE protocol later referred as LSH-3PEKE, in which the server holds a publicly known public key of the client to prevent these attacks. Followed by them were some more protocols: LSSH-3PEKE [Lin-Sun|01], CC-3PEKE [Chang|04], LHL-3PEKE [Lee|04] and three-party EKE [Lee|09]. The Lu-Cao scheme (LC-3PEKE) [Chung|08, Lu|07] focuses on eliminating the overhead caused by the PKI at the cost of more message steps. Chien and Wu [Chien|04] had proposed a 3PEKE protocol with only four steps; however, they did not formally prove the security and optimality. They recently proposed a new protocol while analyzing other protocols [Chien|09]. Lo and Yeh, [Lo-Yeh|09] demonstrated cryptanalysis of two recent 3PEKE protocols proposed by Chen et al. [Chen|08] and Yoon and Yoo [Yoon-Yoo|08] are insecure against the undetectable on-line password guessing attacks.

Sun et al. proposed a new protocol, SCH-3PEKE [Sun-Chen|05], where the server is equipped with a public key and the clients use the public key to securely transmit their passwords and content of communication. This approach is more efficient in terms of the number of message steps and is quite suitable for those environments with a public key infrastructure available.

One of the satisfying outcomes on session key generation is Joux's tripartite key exchange protocol [Joux|03] which makes use of pairings on elliptic curves and requires a single round of communications among parties. Wen

et al. [Wen-Lee|05] also use pairing-based approach, however, their protocol was claimed to be provably secure against active adversaries under a certain intractability assumption.

Despite the fact that the clients have to completely trust the server, the three-party model offers an effective, realistic solution to the problem of session key exchange in large peer to peer systems, and is accepted by the popular Kerberos [Steiner|98] and KryptoKnight [Molva|92] authentication systems.

3.5.1 Kerberos

Kerberos is a secret key based service, to authenticate peers in a network and to distribute session keys between them [Schiller|88]. It is a client-server model which provides mutual authentication, i.e., both the user and the server verify each other's identity. Kerberos makes use of symmetric key cryptography and requires a key distribution centre (KDC). Extensions to Kerberos can provide for the use of public-key cryptography during certain phases of authentication. Kerberos protocol messages are protected against eavesdropping and replay attacks.

KDC can be logically divided into three parts:

- i. *Database*, which is the container for entries associated with users and services. In most cases AS and TGS (they together form the KDC) run on same machine so they both use the same database of secret keys;
- ii. *Authentication Server (AS)* is the part of the KDC which on receiving the authentication request, authenticates the client typically by password based authentication scheme, after which it issues a

special ticket known as the *Ticket Granting Ticket (TGT)* to the client;

- iii. *Ticket Granting Server (TGS)* is the KDC component which distributes service tickets to clients with a valid TGT, guaranteeing the authenticity of the identity for obtaining the requested resource on the application servers.

Application Server (B) is the server that clients want to access for which they must obtain a *service ticket* after acquiring a *ticket*. A ticket is something a client presents to an AS to demonstrate the authenticity of its identity. A service ticket is provided by TGS and a client who has acquired a service ticket is eligible to access the application server.

All the clients share a secret key (generally this secret key is encrypted with a password based encryption algorithm), with the AS and the TGS. Usually, the clients do only know the password to decrypt the secret key. Tickets are issued by the AS and are encrypted using the secret key shared only between the AS and the server providing the service, not even the client which requested the ticket can know it or change its contents.

Obtaining a TGT

AS_REQ is the initial user authentication request directed to authentication server (AS).

$A \rightarrow AS: A$, where A is client A/credentials that identify A.

The AS fetches for K_A which is the key shared between AS and A, from its database. AS_REP is the reply of the AS to AS_REQ that contains the TGT (TGT is the identity of A, S_A , timestamp, validity, etc., all encrypted with

the symmetric key known only to KDC), and S_A (S_A is the session key to be used between AS and A), both encrypted with K_A .

$$AS \rightarrow A: K_A (S_A, TGT),$$

Obtaining a Ticket

Once the client has acquired a ticket it is approved of requesting for a service ticket. TGS_REQ is the request from the client to the TGS for a *service ticket*. It includes the TGT obtained from the AS_REP; and an authenticator (timestamp T_A), generated by the client, both encrypted with the session key S_A .

$$A \rightarrow TGS: TGT, S_A (T_A)$$

The TGS validates the ticket and on successful verification responds with TGS_REP. A TGS_REP consisting of: (a) Identity of application server, (b) a new session key K_{AB} (which is session key to be used between the client A and the application server B), and (c) requested service ticket (which consists of the new session key K_{AB} , timestamp, lifetime, and identity of A, all encrypted with the secret key known only to KDC), all these three components are encrypted by S_A .

$$TGS \rightarrow A: S_A (B, K_{AB}, ticket_B)$$

Login into the Application Server (B)

AP_REQ is the request that the client sends to an application server to access a service. The AP_REQ consists of the *service ticket* obtained from TGS and an authenticator (timestamp T_A generated by the client) encrypted using the service session key (K_{AB}).

$$A \rightarrow B: ticket_B, K_{AB}(T_A)$$

AP_REP is the reply from AS to the client to prove it really is the server the client is expecting. However, it is optional when client requests the server for it, i.e., when mutual authentication is necessary.

$$B \rightarrow A: K_{AB}(T_{A+1})$$

Thus, the Kerberos protocol mutually authenticates client and application server, and assures that the established service session key is only known to them and the KDC. Further, it proves to the client that K_{AB} has been newly generated. However, this is not achieved from application server's point of view, since application server will accept K_{AB} as long as the ticket has not expired. Further, the lack of this proof to application server makes it possible for the client to cache the ticket and use it for future communication within the ticket's validity.

3.6 Key Agreement Protocols based on Symmetric-Key Techniques

The three-party key agreement protocols mediated by the trusted server, where each client shares a secret key with the trusted server are as follows.

3.6.1 ISO/IEC 11770-2

The first edition of the international standard ISO/IEC 11770-2 [ISO96] appeared in 1996 with mechanisms 12 (as shown in Table 6) and mechanism 13 (as shown in Table 7) for key establishment, where two parties who wish to establish a shared secret key already share a secret key with a trusted third party. Later, ISO/IEC 11770-3 [ISO08] standardized seven generic key agreement schemes based on asymmetric techniques.

ISO/IEC 11770-2 protocol assumes that all the three parties have loosely synchronised clocks T_X (or synchronised sequence numbers N_X) and clients shares a long-term secret key with the trusted server (K_{XS}). The client who initiates the communication controls the key generation process and then securely transfers it to the other client with the help of the trusted server. An optional handshake in the protocol is designed to provide entity authentication and key confirmation. The initiator can make use of three different types of time variant parameters (TV P) and protocol steps are shown in Table 6. The mechanism 13 makes use of random numbers instead of time variant parameters as shown in Table 7.

| |
|--|
| <ol style="list-style-type: none"> 1. $A \rightarrow S: \{TV P_A, B, K_{AB}\} K_{AS}$ 2. $S \rightarrow A: \{TV P_A, B\} K_{AS}, \{T_S/N_S, K_{AB}, A\} K_{BS}$ 3. $A \rightarrow B: \{T_S/N_S, K_{AB}, A\} K_{BS}, [\{T_A/N_A, B\} K_{AB}]$ 4. $B \rightarrow A: [\{T_B/N_B, A\} K_{AB}]$ |
| Table 5: ISO/IEC 11770-2 Key Establishment Mechanism 12 |

In 2006 Cheng and Comley [Cheng06] described two attacks (replay attack and type attack) on mechanism 12 from ISO/IEC 11770-2:1996. Thereafter, ISO/IEC SC27 committee released the 2nd edition of ISO/IEC 11770-2 with fixes for these attacks.

| |
|--|
| <ol style="list-style-type: none"> 1. $A \rightarrow B: R_A$, Random number R_A is generated by A 2. $B \rightarrow S: \{R'_B, R_A, A, K_{AB}\} K_{BS}$, Random number R'_B is generated by B 3. $S \rightarrow B: \{R'_B, A\} K_{BS}, \{R_A, K_{AB}, B\} K_{AS}$ 4. $B \rightarrow A: \{R_A, K_{AB}, B\} K_{AS}, \{R_B, R_A\} K_{AB}$ 5. $A \rightarrow B: \{R_B, R_A\} K_{AB}$ |
| Table 6: ISO/IEC 11770-2 Key Establishment Mechanism 13 |

However, in 2008, Mathuria and Sriram described new type-flaw attacks [Mathuria08] on both mechanism 13 and the fixed version of mechanism 12 proposed by Cheng and Comley.

3.6.2 Gong's Alternative Protocol

In this protocol [Gong93] as shown in Table 8, the participating clients A and B provide some random input that is used to create the session key on both sides. Here, the KDC server (S) authenticates the input's origin and delivers it to the intended communication partner. Clients A and B then calculate the session key as $f(k_1, k_2)$, where f is a one-way hash function and k_1 and k_2 are some input from A and B.

- | |
|---|
| <ol style="list-style-type: none"> 1. $A \rightarrow S: A, B, \{A, S, A, k_1, B\}_{K_{AS}}, n_A$ 2. $S \rightarrow B: A, B, \{S, B, A, k_1, B\}_{K_{AS}}, n_A$ 3. $B \rightarrow S: \{B, S, B, k_2, A\}_{K_{BS}}, \{B, A, n_A\}_{K_{AB}}, n_B$ 4. $S \rightarrow A: \{S, A, B, k_2, A\}_{K_{AS}}, \{B, A, n_A\}_{K_{AB}}, n_B$ 5. $A \rightarrow B: \{A, B, n_B\}_{K_{AB}}$ |
|---|

Table 7: Gong's Alternative Protocol

3.7 Key Transport Protocols based on Public Key Encryption

Key transport protocols based on public-key encryption involve one party choosing a symmetric key and transferring it to a second party by encrypting it with other party's public key. This provides entity authentication via public-key decryption, and data origin authentication via digital signatures.

3.7.1 Needham-Schroeder Public-key protocol

The Needham-Schroeder two-party public-key protocol [Needham|78] provides mutual entity authentication and mutual key transport where each party transfers a symmetric key to the other. As shown in Table 9, the transported keys may serve both as nonce (a random number only used once), for entity authentication and secret keys for further use. Each party possess the public key of the other party to whom they wish to communicate.

| | |
|---|--|
| 1. $A \rightarrow B: P_B\{k_1, A\}$ | Upon receiving message, B decrypts it to get k_1 and replies to A by k_1, k_2 encrypted by A's public key. |
| 2. $B \rightarrow A: P_A\{k_1, k_2\}$ | A on receiving the message decrypts it to get k_2 . |
| 3. $A \rightarrow B: P_B\{k_2\}$ | A then replies to B by sending the encrypted k_2 . The session key can be a function of k_1 and $k_2, f(k_1, k_2)$. |
| Table 8: Needham-Schroeder public-key protocol | |

3.7.2 X.509 Strong Authentication Protocol

The X.509 is a certificate based two-party authentication protocol [PKIX] which outlines mutual entity authentication with optional key transport. The protocol consists of two passes and makes use of timestamps and a challenge-response based on random numbers, as shown in Table 10. It ensures data origin authentication, freshness of the message constructed, confidentiality of the messages communicated, and that the message constructed for an entity is indeed specifically intended for it.

| $D_A = (T_A, r_A, B, [data_1], [P_B\{k_1\}]), D_B = (T_B, r_A, r_B, A, [data_2], [P_A\{k_2\}])$ | |
|---|---|
| 1. $A \rightarrow B: CertA, D_A, \{D_A\}S_A$ | A computes D_A and sends to B along with its certificate and signed D_A . |
| 2. $B \rightarrow A: CertB, D_B, \{D_B\}S_B$ | B computes D_B and replies to A in the same format. |
| Table 9: X.509 Two-way authentication protocol | |

3.7.3 SSL/TLS protocol

The Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol [Thomas00] provides an encrypted channel and integrity protected communication between two parties. Authentication is optional and in most cases only used to authenticate the server to the clients. The TLS standard protocol consists of a handshake mechanism, to negotiate the security parameters, and of a message format to exchange the protected payload.

TLS Handshake

The client requesting a TLS connection first sends a *ClientHello* to the server to which server responds with a *ServerHello*. After this message exchange, the protocol version, a session ID, the ciphersuite, the compression method, etc., are agreed. The *ciphersuite* describes the cipher algorithm, the key exchange method, and the hash algorithm for further use.

In Figure 1, the messages in parentheses are optional. These optional messages are sent according to the chosen ciphersuite or the authentication capabilities. For example, if the server has a key that is only usable for signing, the *ServerKeyExchange* message is sent along, which contains the necessary information to perform a Diffie-Hellman key agreement. The

server may request the client's certificate *CertificateRequest* and send the *ServerHelloDone* message afterward.

| Client | Server | Handshake Type |
|---------|--------|--|
| -----> | | ClientHello |
| <----- | | ServerHello (Certificate) (ServerKeyExchange) (CertificateRequest) ServerHelloDone |
| -----> | | (Certificate) ClientKeyExchange (CertificateVerify) ChangeCipherSpec Finished |
| <----- | | ChangeCipherSpec Finished |
| <-----> | | Application Data |

Figure 1: TLS Hand shake

The client can now verify the signature of the server's certificate. However, the verification is not a part of the TLS protocol and so it is usually done in the application layer. The *ClientKeyExchange* message contains either the encrypted premaster secret (or ephemeral key) or its Diffie-Hellman parameters. *CertificateVerify* message verifies the validity of the client's certificate which actually proves that it knows its private key. The *ChangeCipherSpec* message indicates that all the necessary security parameters have been received on this side, and that the other party may

now use the negotiated master secrets² in the protocol messages as mentioned in Figure 1.

3.8 Key Transport Protocols

When a key transport protocol makes use of both asymmetric and symmetric techniques they are called hybrid protocols such as the Beller-Yacobi protocol.

3.8.1 Beller-Yacobi Protocol

The key transport protocol of Beller and Yacobi is available in two variants 4-pass (as shown in Table 11) and 2-pass, which provide mutual entity authentication and explicit key authentication. This two-party protocol was specifically designed to minimize the computational requirements. The 2-pass protocol has slightly weaker assurances for entity authentication and key authentication. However, for an entity to obtain explicit key authentication a third message could be added by means of challenge and response technique.

| | |
|--|--|
| 1. $A \rightarrow B: A, P_A, \text{Cert}A$ | A sends its public key and certificate. |
| 2. $B \rightarrow A: \{X\}P_A$ | B encrypts a random X with A's public key. |
| 3. $A \rightarrow B: \{N_A\}X$ | A encrypts a random nonce with X. |
| 4. $B \rightarrow A: \{B, P_B, \text{Cert}B, \{N_A\}S_B\}$ | B sends its cert with signed nonce send earlier. |
| Table 10: Beller-Yacobi 4-pass key transport protocol | |

² The Pseudo Random Function (PRF) is an iterative function that produces more output the more iteration it goes through. The *master secret* is generated using PRF with inputs such as, premaster secret and some random inputs from Client and the Server.

3.9 Key Agreement Protocols based on Asymmetric Key Techniques

The key agreement protocols that make use of the asymmetric techniques for entity authentication and key transport are as follows.

3.9.1 Diffie-Hellman Key Agreement

Diffie-Hellman (DH) is a basic protocol that provides unauthenticated key agreement in a distributed unsafe network. The security of DH protocol relies on the intractability of the Diffie-Hellman problem and the related problem of computing discrete logarithms.

| | |
|---|---|
| 1. $A \rightarrow B: m^x \bmod p$ | A generates a random x ($1 < x < p-2$) and computes m^x where m is the generator of the cyclic group. |
| 2. $B \rightarrow A: m^y \bmod p$ | B computes m^y where ($1 < y < p-2$) and replies to A. |
| A and B compute $(m^x)^y$ and $(m^y)^x$ respectively, at their ends which is the session key. | |
| Table 11: Basic Diffie-Hellman Key Agreement | |

In a two-party public key based key agreement it is assumed that the clients know the public key of the other clients with whom they want to communicate securely, as shown in Table 12. This means that either, each client needs to store public keys of all the clients with which they wish to communicate, which of course is a significant memory overhead, or they have to trust a client claiming his public key to be authentic. In that case, the clients have to make an extra effort for certificate verification of the other

client from either the published CRL (Certificate Revocation List) or an OCSP (Online Certificate Status Protocol) [RFC 2560], for each transaction which incurs transmission overhead. Considering this, it would be convenient to have a trusted server in the system that does the job of authenticating the clients and establishing the session key between them.

3.10 Authentication in Three Party Key Agreement Protocols

Informally, the authentication in a tripartite (three-party) key agreement can be divided in two categories:

(a) Certification-based protocols [Riyami|02, Riyami|03, Cheng|04, Joux|03, Shim|03, Aivaloglou|08 and Buttyán|10] use certificates issued by a certification authority, to bind a client's identity with his public keys.

(b) ID-based protocols³ [Chien|04, Liu-Zhang|03, Nalla|03, Nalla|03, Shim|03b, Zhang-Liu-Kim|02, Tso|05, Boneh|01, Smart|02, McCullagh|05, Wang|05, Cheng|05, Sakai|00 and Gorantla|08], where user's public key is an easily calculated function of her identity (e.g., social security number), while the user's private key is calculated by a KGC (Key Generation Centre). Shamir was the first to introduce the concept of an identity-based cryptosystem [Shamir|85].

An alternative to public-key certificates and identity-based systems is *implicitly certified public keys* [Menezes|96] where entity's public data (which replaces certificate) consists of trusted party related public data,

³ When clients use their identity-based asymmetric key pairs for authentication and to establish key, instead of using traditional public/private key pairs, then the authenticated key agreement protocol is called identity-based.

entity's identity, and additional per-user public data. However, the integrity of public key is not directly verifiable but a valid public key can be recovered only from the public data of an authentic user.

The two classes of implicitly-certified public keys are: *Identity-based public keys*, where private key of each entity is computed by the trusted party and securely transferred to the entity; and *self-certified public keys* where each entity computes its private key and corresponding public key.

Marko et al. [Hölbl09] analyzed tripartite authenticated identity-based key agreement protocols using pairing-based key agreement protocol and conducted a comparative study of each protocol on the basis of security properties, attacks and the computational effort required. They also suggested that *Identity-based authenticated key agreement protocols can be an alternative for certificate-based protocols when efficient key management and moderate security is required*. Finally, they concluded that Shim's protocol [Shim03] is efficient and at the same time offers all security properties. Wanga et al. [Wanga09] analyzed various identity-based key establishment protocols: Chen-Kudla [Chen02], Smart's [Smart02], Wang's [Wang05], Chow-Choo [Chow05], SYL [Chen06], against reflection attack, provable security, modular proof, etc., and discovered another attack called PKG forward secrecy. Further, they proposed a new protocol RYY⁺ which is resilient to it.

3.11 Properties of Key Establishment Protocols

There are some generic properties that we seek in key establishment protocols as advised in [Menezes96, Eddie05]. Typically, any

authentication and key establishment protocol is evaluated using the properties listed in Table 13. Our motive here is to know all these properties before we evaluate the AK-protocol.

| | |
|--|--|
| <i>Communications efficiency</i> | It is desirable for a protocol to have a small number of message exchanges (or passes) between parties and a low bandwidth. |
| <i>Composition of secure channels</i> | The entire process of key establishment and the resulting session key are sufficient to guarantee the composition of the secure channel. |
| <i>Computational efficiency</i> | This property is of major concern in wireless devices where low computational complexity is desired. |
| <i>Data Confidentiality</i> | Secret keys and possibly other data are to be kept confidential while being transmitted or stored. |
| <i>Efficiency</i> | Considerations include the number of message exchanges (passes) required, <i>message rounds</i> ⁴ required, the number of bits transmitted and the complexity of computations required by each party. |
| <i>Entity Authentication</i> | Mutual authentication provides both entities with assurance of each other's identity while in the case of unilateral authentication only one entity is provided with such assurance. |
| <i>Ephemeral</i> ⁵ <i>key</i> | Compromise of ephemeral secrets does not reveal long- |

⁴ Message round: The integration of one or more message steps if there is no data dependency between these steps. These steps can be executed in parallel to save communication time.

⁵ Key establishment protocol entails three kinds of keys: ephemeral keys, session keys or long-term keys. Ephemeral keys are temporary secret values that are needed during the execution of a key establishment protocol. Session keys are keys that are computed at the

| | |
|-------------------------------------|--|
| <i>security</i> | term secrets or session keys. |
| <i>Explicit key Authentication.</i> | A protocol that provides both implicit key authentication and key confirmation is said to possess explicit key authentication. |
| <i>Good Key</i> | A key is good if it is fresh and authenticated. |
| <i>Implicit key authentication</i> | An agreed-upon secret key should be known only by identified parties. |
| <i>Key Authentication</i> | The key is known only to the participating clients and any mutually trusted parties. |
| <i>Key compromise impersonation</i> | Compromise of an entity long-term private key should not enable the adversary to impersonate as him. |
| <i>Key Confirmation</i> | The assurance for one entity that another identified entity is in possession of the correct key. |
| <i>Key Control</i> | Neither entity should be able to force the session key to be a preselected value. |
| <i>Key Freshness</i> | Guarantees that the established keying material is new, as opposed to the reuse of old keying material. |
| <i>Key integrity</i> | This property assures that the key has not been modified or disturbed by an adversary. |
| <i>Known-key security</i> | Each run of the protocol should result in a unique secret session key. The compromise of one session key should not compromise other session keys. |
| <i>Non-repudiation</i> | When a key establishment protocol prevents any party |

end of a key establishment protocol's execution which is valid during the session they were created for. Long-term keys are values such as the private keys of the parties or the master key that are saved in a trusted server.

from denying having previously established a session with another party.

*Perfect Forward
Secrecy*

A key establishment protocol is said to offer perfect forward secrecy if compromise (or release) of long-term keys does not compromise past session keys.

*Resistance to known
attacks*

The key establishment protocols are desired to survive typical attacks like man-in-the-middle, reflection and interleaving attacks.

*Unknown key-share
resilience*

An adversary should not be able to forge or trick a benign entity to share his secret key with him.

Chapter 4 Public Key Infrastructure

4.1 Chapter Introduction

This chapter assembles all the relevant details required to understand the Public Key Infrastructure (PKI), the functioning of certification and registration authority, directory service, certificate issuance, verification and revocation, various modes of certification and X.509 certificates.

4.2 Public Key Infrastructure

Prior to exchange of any crucial information, users of a system must be assured of identity of the other communicating users. Confidentiality of the information and confidence in the integrity of exchanged information is critical. This list of security services can be further extended to non-repudiation of agreements, digital notarization and securely time-stamping the transactions of information. PKI provides a long-term, well-conceived infrastructure to efficiently deliver these services in a cohesive manner.

Although symmetric cryptography was computationally efficient, it suffered from the fact that it could not support certain security services and it presented a difficult key management. However, when Diffie and Hellman

first introduced public key cryptography the whole focal point changed [Diffie-Hellman|76].

Public key cryptography supports security mechanisms such as confidentiality, integrity, authentication and non-repudiation. Public key cryptography is based on the use of *key pairs (Public and Private Key)*. Private-key must be kept secret and is under the control of the owner. Public-key, can be available to other users who would like to establish any security services, with the person holding the private key. This is possible because the keys in the pair are mathematically related but it remains computationally infeasible to derive the private key from knowledge of the public key.

The purpose of a public key infrastructure (PKI) framework is to enable and support the secured exchange of data, credentials in insecure environments. Assuming that “there always exist an insecure channel between pair of users at any instance of time” [Maurer|94]. A PKI is a foundation on which other applications, system, and network security components are built which consists of security and operational policies, security services, and interoperability protocols supporting the use of public-key cryptography for the management of keys and certificates.

4.3 Certification & Registration Authority, and Certificate Directory

Certification Authority (CA) functions as a trusted third party and provides various key management services. A CA essentially certifies the identity of an end entity. This is accomplished by an entity providing sufficient proof

of their identity to the CA. The key management-related functions performed by a CA are: *Certificate generation* and *Certificate revocation*.

A *Registration Authority (RA)* is an optional but common component of a PKI who performs some of the administrative tasks that a CA would normally undertake. The primary purpose of an RA is to verify an end entity's identity and determine if an end entity is entitled to have a public key certificate issued. The RA must enforce all policies and procedures defined by a CA.

Public key certificates are issued for a fixed period of time before they become void; situations can arise where they are no longer trustworthy and thus must be prematurely expired known as *Certificate Revocation*. A *Certificate Revocation List (CRL)* or *Certificate Directory (CD)* is a list generated by the CA that contains unique information about the revoked certificates which enables relying entities to determine if a certificate is valid or not. A CRL entry is serialized, time-stamped, and signed by the CA and published in a publicly available repository or directory. This service of publishing revoked certificates provided by CA is known as *Directory Service*.

4.4 Digital Certificate

A Public Key Infrastructure is designed to provide the trust using a data element called a *digital certificate* or *public key certificate*, which binds a public key to identifying information about its owner. The infrastructure is designed to create the binding, and manage it for the benefit of all within the community of use. The generation, distribution, and management of public keys and associated certificates normally takes place by the use of CA, RA, and Directory Services, which can be used to establish a hierarchy or chain

of trust. They allow for the implementation of digital certificates that can be used to identify different entities.

4.5 PKI Functions

PKI functions include:

- (a) *Public key cryptography*, which includes the generation, distribution, administration, and control of cryptographic keys.
- (b) *Issuing certificates*, binding a public-key to an individual, or an organization.
- (c) *Certificate validation*, verifies the existence of a trust relationship and that a certificate is still valid for specific operations.
- (d) *Certificate revocation*, withdrawal of previously issued certificates that have become invalid due to some reasons and either publishing it to a *Certificate Revocation List (CRL)* [RFC 3280] or enabling an *Online Certificate Status Protocol (OCSP)* [RFC 2560] process, to be accessible publicly.

When building a comprehensive infrastructure, there are additional management functions that might be required in certain environments as recognized by PKIX working group. The PKIX Certificate Management Protocols [RFC 2510] provides the most complete set of management functions that might be required in a comprehensive PKI.

4.5.1 Certificate Binding or Certificate Issuance

A client who wants to make use of PKI services must first, enrol or register himself to a CA. Registration process includes the process of generation of

public and private keys. Then this public key along with the credentials that validate the identity of the clients is submitted to a CA, which is later verified by the RA. The primary objective of a CA is to bind the identifying information and credentials supplied by a client with the public key in the form of a certificate. The binding is declared when a trusted CA digitally signs the public key certificate of the client with its private key. Otherwise, an impostor can replace the public key in a certificate with its own public key and can impersonate the true application and gain access to secure data.

4.5.2 Certificate Verification

When an entity wishes to obtain a certificate for verification purposes, the certificate may be retrieved from a directory service, a local cache, or an OCSP authentication message. If certificates are retrieved from a directory service or a local cache, it is most expedient to do this prior to the authentication exchange. During an authentication exchange, the entity generating the request may send its certificate or a chain of certificates with the request, which are validated and the signature of the CA is verified. Certificate formats are outside the scope of this section, but particular formats may be recommended or mandated by other FIPS, NIST Special Publications, or other Federal regulations.

4.5.3 Certificate Revocation

Public key certificates are issued with fairly generous lifetimes. However, the circumstances that existed when the certificate was issued can change before the certificate would naturally expire. Reasons for revocation include private key compromise, change in affiliation, name change, etc., (specific

reason codes are defined in X.509). Therefore, it is sometimes necessary to revoke a certificate before its expiration date.

The Revocation Request allows a RA to request revocation of a given certificate. Certificate revocation information must be made available by the CA who issued that certificate or by the CRL Issuer to which the CA delegates this function. X.509 defines a method for publishing this information via Certificate Revocation Lists (CRLs). The frequency of publication and the type of CRLs used are a function of local policy.

4.6 Hierarchical and Non-Hierarchical Certification

A *hierarchical public key certification* typically is a simple model which allows certificates of the users to be signed by a single CA. The hierarchy consists of a series of CAs that are arranged based on a predetermined set of rules and conventions. In a *non-hierarchical public key certification*, no trusted third party actually vouches for the identity or integrity of any end entity. Pretty Good Privacy (PGP) [Zimmermann|95] uses this type of trust model in email environments.

4.7 Certificate Chain

A *certification path* is a chain of certificates between any given certificate and its trust anchor (CA). Each certificate in the chain must be verifiable in order to validate the certificate at the end of the path. This functionality is critical to the usability of PKI [CPC|02].

4.8 Cross Certification

A *Cross-certificate* is a public key certificate that is issued by one CA to another CA. In other words, a cross-certificate is a public key certificate that contains the public key of a CA that has been digitally signed by another CA. Cross-certification can be bi-directional or unidirectional. *Bi-directional cross-certification* typically occurs between peer CAs. *Unidirectional cross-certification* typically occurs in a hierarchical trust model where superior CAs issue cross-certificates to subordinate CAs, but the reverse is not true.

4.9 X.509 Certificates

The PKIX Working Group was established in the fall of 1995 with the goal of developing Internet standards to support *X.509-based PKI* [PKIX].

4.9.1 Contents of X.509 Certificates

An X.509 [X.509] certificate⁶ (filename extension) contains information about the certificate subject and the certificate issuer (the CA that issued the certificate). A sample self-signed certificate (a certificate signed by the owner himself) is shown in Figure 2.

⁶ Common filename extensions for X.509 certificates are:

- .pem - (Privacy Enhanced Mail) Base64 encoded DER certificate, enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----"
- .cer, .crt, .der - usually in binary DER form, but Base64-encoded certificates are common too (see .pem above)
- .p7b, .p7c - PKCS#7 Signed-Data structure without data, just certificate(s) or CRL(s)
- .p12 - PKCS#12, may contain certificate(s) (public) and private keys (password protected)
- .pfx - PFX, predecessor of PKCS#12 (but most of the time the actual data is PKCS#12, only the extension is kept)

A certificate is encoded in *Abstract Syntax Notation One (ASN.1)*, a standard syntax for describing messages that can be sent or received on a network. A certificate includes:

- A subject *Distinguished Name (DN)* that identifies the certificate owner.
- The public key associated with the subject.
- X.509 version information.
- A serial number that uniquely identifies the certificate.
- An issuer DN that identifies the CA that issued the certificate.
- The digital signature of the issuer.
- Information about the algorithm used to sign the certificate.
- Some optional X.509 v.3 extensions, e.g., an extension exist that distinguishes between CA certificates and end-entity certificates.

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=NZ, ST=Dunedin, L=University of Otago, O=Information
Science,
    OU=Certification Services Division,
    CN=UnivOtago CA/emailAddress=server-certs@Otago.com
  Validity
    Not Before: Aug 1 00:00:00 1996 GMT
    Not After : Dec 31 23:59:59 2020 GMT
    Subject: C=NZ, ST=Dunedin, L=University of Otago, O=Information
Science,
    OU=Certification Services Division,
    CN=UnivOtago CA/emailAddress=server-certs@Otago.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
    Modulus (1024 bit):
      00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c:
      68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da:
      85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06:
      6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2:
      6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b:
      29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90:
      6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f:
      5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36:
      3a:c2:b5:66:22:12:d6:87:0d
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints: critical
    CA:TRUE
  Signature Algorithm: md5WithRSAEncryption
    07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9:
    a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48:
    3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88:
    4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9:
    8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5:
    e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9:
    b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e:
    70:47
  
```

Figure 2: Sample Self-Signed Certificate

Chapter 5 AK-Protocol

Description

5.1 Chapter Introduction

In this chapter we illustrate in detail the AK-protocol for distributed and closed environment, its system setup requirements, key generation and mathematical computation required at server and client end. We also discuss the properties exhibited by the AK-protocol such as perfect forward secrecy, known session-key security, unknown key-share resilience, key-control, desired properties in 3PAKE, non-repudiation, key escrow, key freshness, key compromise resilience, scalability, required bandwidth, centralised key distribution and directory service. Thereafter, we analyse the AK-protocol for bit complexity, message flow, efforts required for its cryptanalysis, including the system under different attack situations such as replay, impersonation, DDoS and a special situation where an attacker can craft protocol messages to mislead the clients.

5.2 The Basis of AK-protocol

In previous chapter, we have seen different kinds of key establishment protocols which either require a trusted third party or not to mediate the communication. Specifically, in a three-party based key establishment protocols, authentication is usually accomplished by the trusted server, while for session key generation is mostly controlled by the clients, i.e., the keying material, or a part of the key is generated at one of the clients. But in this thesis we are proposing a new scheme where trusted server controls the generation and distribution of ephemeral keys, after which the participating clients exchange their ephemeral keys to generate the session key at both ends. A trusted server controlled approach was proposed by Guo et al. [Guo|09] for threshold identity-based authenticated server-controlled gateway-user key exchange. However, in our protocol trusted server besides authenticating the clients also oversees the generation and distribution of the ephemeral keys. After which, the clients have to exchange the ephemeral key they possess with the other participating client to establish a (common) session key at both ends. Thus, the session key generation requires participation of trusted server and both participating clients.

In our protocol we have incorporated the advantages of:

- a) Certificate-based authentication which is typically used where high level of security is required. Since certificates are a reusable component that provides strong authentication. They also offer solutions to scalability and mobility to individual clients and the entire system.
- b) Trusted server based authentication, where a centralized system which is well known to all the clients in the system authenticates the participating

clients. This is considered more reliable than peer-to-peer authentication specifically for high-valued information exchange.

- c) Trusted server controlled process of key establishment, where a trusted server not only authenticates the participating clients but also generates and distributes the ephemeral keys to them.

This approach brings some added capabilities to the system like, Non-repudiation, Key escrow, ease of key management, Key freshness by elements other than time-stamp, Integrity of the ephemeral keys and the session key, Explicit and Implicit key authentication, Unknown-key share resilience, Control of the session key generation remains equally with the trusted server and the participating clients, Entity authentication, and mitigation of various conventional attacks like, man-in-the-middle, reflection, flooding, and interleaving attacks.

5.3 The AK-protocol Overview

The AK-protocol establishes a key between two clients that can be used to have a secured communication between them while making use of a trusted server to mediate the process of authenticating the participating clients and key establishment. The outcome of the protocol is a *session key* which can be used to secure further communications between the two clients. There can be a pre-negotiated asymmetric or symmetric key encryption/decryption algorithm (like AES, DES, RSA, ECC, etc.) in the system or between the participating clients, which can be used to encrypt/decrypt the communications, after the clients have established a common session key.

The architecture of the protocol requires a trusted server, which is known to all the clients in a system, to mediate the process of key establishment as shown in Figure 3. The protocol accomplishes the process of key establishment in two phases:

- I. **Entity Authentication**, two clients wanting to have a shared secret (session) key should first authenticate themselves to the trusted server. This may vary in different kinds of environments: distributed, closed and trusted, for example:
 - a. In a large distributed environment where clients can be identified by the trusted server using a certificate-based or identity-based authentication schemes;
 - b. In a closed environment where trusted server is informed about each client in the system, self-signed certificate-based or password-based authentication can be used; and
 - c. In a trusted environment where trusted server and the clients are well-informed of the other participating client, key pairs (public and private) can be used to bind their identities.

For a comprehensive understanding of the AK-protocol we pick an example of a distributed system where certificates are used as a mode of authenticating each client, prior to session key-establishment. The client authentication as per the protocol is a three step process as described in Section 5.5, Phase 1.

- II. **Session Key Establishment** is a two step process. From a trusted server's perspective, after clients are authenticated successfully they are eligible to access the keying material or ephemeral keys.

First, trusted server creates two *different but related* parts of the keying material (*key-parts* or *ephemeral keys*) and encrypts them with the public keys of the respective clients; and sends these two encrypted key-parts called *protocol messages* (or just *messages*) along with the public key of the other client to them. When clients receive their encrypted key-part (or message) they decrypt it using their private key to retrieve the key-part. After trusted server distributes the key-parts it may disconnect from these two clients specifically in a closed or trusted system, while the participating clients can continue to have their communication until they are done.

Second, at this stage each participating client has a different key-part. The clients encrypt the key-part they have, with the public key of the other client to create another message. This message is now ready to be exchanged with the other client. At the end of this exchange, both the clients have two key-parts. The clients have to compute modular-multiplication of key-parts to obtain a *session key* which is used to encrypt further communications between these two participating clients.

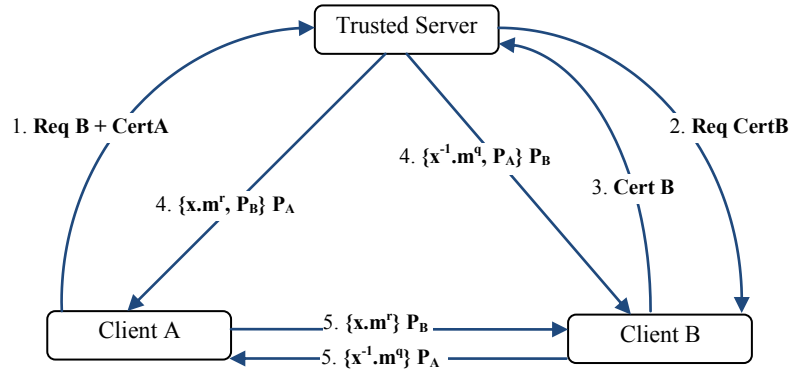


Figure 3: The Protocol Overview

5.3.1 System Setup and Key Generation

At the time of initializing the system,

- i. A secured database is created within the trusted server which will later store the public-key certificates of the subscribed clients, and information incorporating their identities. Optionally, there can be another database to store the keying material used for each transaction;
- ii. A set of applications are built-in the server that would serve the functions of public key cryptosystem, e.g., encryption/decryption, certificate verification and random number generation;
- iii. A pool of pairs of multiplicative inverses over modulo operation N (which is different from modulo operator $\{\eta\}$ used for key generation) is constructed to reduce the effort of generating a pair of multiplicative inverses for each transaction.

Each new client joining in the system generates a long term public-private keys using any of the public key cryptosystems⁷ like, RSA, ElGamal, Rabin, ECC [Yang-Chang|09]. However, we believe that in the near future well-known and time-tested RSA cryptosystem will continue to dominate. Therefore, we prefer RSA as the algorithm for key generation and encryption/decryption in the rest of this thesis.

The client (which has generated its key pairs), follows the certification process (see Section 4.5.1 for more information) to generate a public-key certificate which is similar to X.509 certificates (public-key of X $\{P_X\}$, modulo $\{\eta\}$, some unique identification $\{I\}$) (see Section 4.7 for more information). This assumes that the client is not providing fake identities or there is a *Registration Authority* (RA) included in the certification process to verify the identities and credentials submitted by the clients. It is also deduced that the liability of the private key lies with the client, since according to the architecture, no form of private key is ever sent over the network under any circumstances. A new pair of keys can be generated by the client in the event of realization of key compromise.

After RA has verified the client's identity successfully, it is provided with a certificate by a single or chain of *Certification Authority* (CA) (see Section 4.5 for more information). Usually, in practice X.509 certificates are used as standard which makes use of RSA public-key algorithm for encryption and decryption.

In a case when public-private key pairs and certificates are issued by a CA, it is transferred securely by means of secondary channel, e.g., courier. The

⁷ In a public-key encryption scheme, the communication partners do not share a secret key. Each user has a pair of keys: a secret key known only to him and a public key known to others who want to communicate.

liability of the credentials now rests with the issuer (CA) and the client. The CA may store the private keys of the clients for the purpose of key escrow and non-repudiation that can be accessed by a law enforcing agency when warranted. The trusted server may store the transaction details referenced with a transaction ID in its database that would facilitate non-repudiation and to resolve any disputes.

Note: Trusted Server is not involved in the process of key generation. CA transfers the database of the Clients' information, e.g., unique ID and Certificates, to the Trusted Server for the purpose of authenticating clients.

5.4 Roles of TS, CA and RA

As per the architecture of the AK-protocol there are two sets of keys: long-term key pairs and short-term or ephemeral keys (key-parts). Long-term key pairs (public and private) are used for encryption/decryption of the protocol messages, and are generated by either CA or the client itself which is secured within their domains, i.e., either client or CA or both. The CA then transfers the long-term public keys of the clients to the TS, hence TS does not hold the private keys of the clients.

Short-term or ephemeral keys are generated and delivered by the TS to the participating clients for them to establish a common session key. The TS stores in its secured database the details of the communication or transaction (e.g., IP of the participating clients, their identity/certificates, date-time, etc.) between certain clients for future references. However, the TS may or may not save the encrypted key-parts or ephemeral keys, which is discussed further in Section 5.8.11.

The trusted server (TS) is a third party that authenticates the participating clients and mediates the process of ephemeral key generation for them to establish a session key. Its role remains the same in different types of environments (Distributed, Closed and Trusted). However, distinctive roles played by the CA and RA depend upon the type of environment.

Generally, in a large distributive network there can be a hierarchy of CAs (Intermediate CAs' and Root CA) that issued a certificate, i.e., the issued certificate contains the certificates of all the intermediate CAs' and Root CA⁸. A Root CA is a well-known trusted party which occupies top most level in the trust hierarchy and its certificate is available to all the registered users (clients). The primary goals of a CA remain the same, i.e., certificate generation and certificate revocation (see Section 4.5.3 for more information). As per the AK-protocol, the process of key generation may or may not be at the CA's end. In case when the client generates pair of keys and the system desires non-repudiation, then the client needs to transfer its key-pairs to the CA's database via secondary channel, e.g., courier.

The specific role played by an RA is to verify the client's identity and attest the credentials provided by them. However, RA may only be required in a distributed environment while in closed and trusted environment it is merged within the CA. Hence, in a closed and trusted environment the CA is also responsible for identifying the clients and confirming their supporting documents, besides its basic job of certificate generation and certificate revocation.

⁸ A Certification Authority's (CA's) public key is distributed to all entities that trust the CA's certificates. If a CA is a Root CA then it must distribute its public keys as self-signed certificates with an acceptable key certificate format and distribution protocol. The CA must also make its public keys available, so that relying entities can resolve the self-signed certificates.

5.5 The Process of Authentication and Key Establishment

In this section we describe the process of authenticating the participating clients and an optional two-way authentication of the trusted server, in phase 1. In phase 2 the trusted server mediates process of key establishment between these two authenticated clients by generating and distributing the ephemeral keys to the clients. After which clients exchange the ephemeral keys they possess to complete the process of key establishment.

Phase 1

Certificate-based Authentication of Both Parties

As shown in Figure 3, before the process of key establishment begins, the participating clients are required to present their certificates to the trusted server (TS) for verification (steps 1, 2 and 3). Hence, the clients are authenticated before being qualified to communicate with the other clients registered in the system, which ensures *mutual authentication* between the clients via trusted server. Optionally, the trusted server can also be authenticated, for instance in a distributed network clients also want to be assured that the trusted server is actually what it claims to be. In such case where *two-way authentication* is required, TS also acquires a pair of keys and a certificate (CertTS) from the CA during system initialization. On request TS presents its certificate to the client for verification.

Step 1: $A \rightarrow TS$: ReqB , CertA

In the first protocol message (see Figure 3), client A requests TS to initiate a communication with client B along with its certificate for authentication. During the certification process, clients are provided with certificates only after the registration authorities (RA) have identified the clients and verified the credentials provided by them. Subsequently, a client possessing a certificate implies that it is identified by the system.

Hence, *entity authentication* is achieved when a client presents its certificate to the trusted server and it is successfully verified and validated. Upon receiving (a) *request to communicate with client B*, TS initiates a communication with client B; and (b) *the certificate from client A*, TS validates and verifies the certificate after which client A is deemed to be authenticated.

Step 2: $TS \rightarrow B$: Req CertB

After establishing a connection with client B, TS requests it to send across its certificate for authentication. So, both the clients are authenticated before they can receive their key-parts from the TS.

Step 3: $B \rightarrow TS$: CertB

After client B presents its certificate to TS for authentication it is validated and verified, and with successful authentication (of client B) both the clients are said to be identified by the system. Optionally, if the system requires two-way authentication, i.e., both trusted server and the participating clients are authenticated by each other while typically only the clients are authenticated by the trusted server. In the case of two-way authentication trusted server presents its certificate for verification to both the participating

clients. Now, all the parties are authenticated before they proceed to the next phase.

Phase 2

Session Key Establishment

In this phase, the participating authenticated clients are provided with the key-parts (ephemeral keys) generated and distributed securely by the TS. Each client present in the system believes that the knowledge of the ephemeral keys lies among the trusted server and both participating clients, and no one else consequently, offering *Implicit Key Authentication*.

By the end of this phase, each participating client is assured that the identified other client possesses the correct key, hence accomplishing *Key Confirmation* and eventually *Explicit Key Authentication* (as per the property of explicit key authentication defined in Section 3.11).

The two different but related key-parts are generated by the TS, which when delivered to the respective clients can be removed from the buffers of the trusted server. This way TS may not need to save the state of the connections and the respective ephemeral keys used for a particular session, making the system *stateless*. However, this also means that we cannot refer to the session keys in the event of any disputes or repudiation.

Nevertheless, if the system demands to offer *non-repudiation*, TS may store the encrypted key-parts in a secured database where a secured access is practiced by TS, which is discussed more in Section 5.8.11. It is up to the system's requirement while initializing the system to make a desired trade-off. Because, we understand that storing the session keys (although encrypted) would also mean that they can be retrieved, later.

Step 4: TS \rightarrow A: $\{x.m^r \pmod{N}, P_B\}$ P_A and

TS \rightarrow B: $\{x^{-1}.m^q \pmod{N}, P_A\}$ P_B

The TS performs the following operations to generate two distinct but related ephemeral keys:

- A. Generate two random numbers *exponents* r and q , where $1 < r, q < N-2$. It is to be noted that in the protocol we are dealing with two different modulo operations. Modulo η is used for key generation and modulo N is used to compute the ephemeral keys, where $\eta > N$. To keep the Figure 3 simple we have excluded the modular function \pmod{N} in the figure.
- B. Randomly choose a pair of integers (x, x^{-1}) which are multiplicative inverses of each other over modulo N , where $1 < x < N-2$. To reduce the computation of generating pairs of multiplicative inverses in real-time, trusted server can generate pairs of multiplicative inverses (excluding pairs which are self inverses) over modulo N during system initialization. These pairs of pre-computed multiplicative inverses can be stored in a *pool of inverses* (POI) for quick access.
- C. Computes two protocol messages for the clients:
 - a. Protocol message for client A:
 - i. Compute $x.m^r \pmod{N}$, where m is the generator of the multiplicative group Z_N^* ,
 - ii. Append the public key of client B (P_B) and,
 - iii. Encrypt these two components with public key of client A (P_A).

b. Protocol message for client B:

- i. Compute $x^{-1}.m^q \pmod{N}$, where m is the generator of the multiplicative group Z_N^* ,
- ii. Append the public key of client A (P_A) and,
- iii. Encrypt these two components using public key of client B (P_B).

D. Distributes the two generated messages to the respective clients. If the system stipulates to store these messages in the secured database (of TS) indexed with a unique transaction ID, it is done now. These stored messages can be used later to assist in settling disputes.

Step 5: $A \rightarrow B: \{x.m^r \pmod{N}\} P_B$ and

$B \rightarrow A: \{x^{-1}.m^q \pmod{N}\} P_A$

Upon receiving their respective protocol messages clients A and B decrypt it with their *private keys* to obtain the two components: public key of the other client and their respective key-part ($x.m^r$ or $x^{-1}.m^q$). They then encrypt the key-part they have received with the public key of the other client. Thus client B receives $(x.m^r)P_B$ and clients A $(x^{-1}.m^q) P_A$ and send it to client B and A, respectively. At the end of this message exchange clients A and B possess both key-parts $x.m^r$ and $x^{-1}.m^q$, which when multiplied over modulo operation N establish the session key at both ends.

The modular multiplication of the key-parts $x.m^r$ and $x^{-1}.m^q$ to generate a session key is shown below.

$$\text{Generated Session Key} = (x.m^r) . (x^{-1}.m^q) \pmod{N}$$

$$= (x \cdot x^{-1}) \cdot (m^r \cdot m^q) \bmod N$$

$$= m^{r+q} \bmod N$$

Note: The generated session key is not saved at the clients' end and is removed from the buffers after its use. However, clients save the transaction ID for future reference.

5.5.1 Distributed Environment with Multiple TTPs

Typically, in a very large distributed environment there is not one but n number of Trusted Third Parties (TTP) that authenticates the clients subscribed in their domain. In this situation the AK-protocol may not vary in terms of ephemeral key generation or the authentication process but there will be added inter-communications between the TTPs.

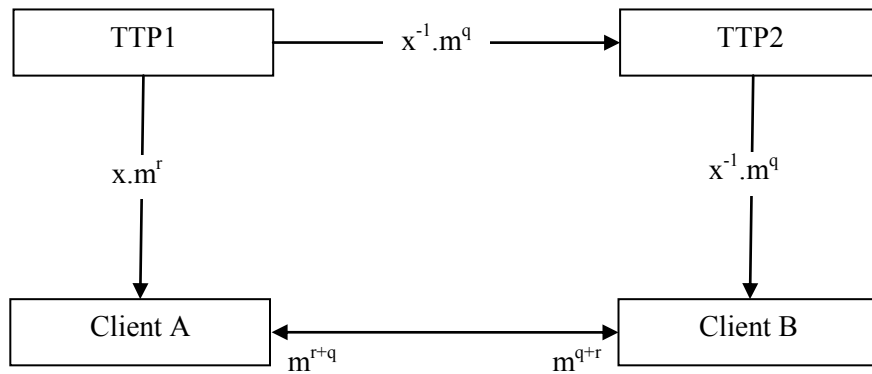


Figure 4: Distributed Environment with Multiple TTPs

For instance, A and B want to communicate securely, where A is subscribed under the domain of TTP1 while B is subscribed under the domain of TTP2.

Client A requests TTP1 to initiate communication with client B. TTP1 authenticates A, and after identifying TTP2 it forwards the request to TTP2. TTP1 needs to have a directory service where it can look up for which clients are subscribed under which TTP. On receiving the request, TTP2 requests client B to present its certificate. Upon successful authentication of client B, TTP2 sends a positive response to TTP1 with the public key of client B. As shown in Figure 4, TTP1 generates the key-parts and encrypts them with respective public keys, which it delivers to client A and TTP2. TTP2 forwards the encrypted key-part to client B. Now both clients can exchange their messages to finish the key establishment process.

5.6 Mathematical Computations

The mathematical computations [Appendix A] that take place on the trusted server and the participating clients during the process of key establishment are shown in Table 13.

| <i>Trusted Server</i> |
|---|
| <ul style="list-style-type: none"> ▪ <i>Generate random numbers r, q such that, $1 < r, q < N-2$.</i> ▪ <i>Compute two key-parts $x.m^r$, and $x^{-1}.m^q$ from the chosen pair of multiplicative inverses (x, x^{-1}), where m is the generator of the multiplicative group Z_N^*.</i> ▪ <i>Perform RSA encryption of the two key-parts. If public key of client A is (e_1, η), and B is (e_2, η). Then encryption of key-parts for the clients A and B is: $(x.m^r, e_2)^{e_1} \bmod \eta$, and $(x^{-1}.m^q, e_1)^{e_2} \bmod \eta$, respectively.</i> |

| <i>Client A</i> | <i>Client B</i> |
|--|---|
| <i>Private key = (d_1, η)</i> <i>Decrypt the received key-part</i> $= (x.m^r, e_2)^{e_1.d_1} \bmod \eta$ $= (x.m^r, e_2) \bmod \eta$ | <i>Private key = (d_2, η)</i> <i>Decrypt the received key-part</i> $= (x^{-1}.m^q, e_1)^{e_2.d_2} \bmod \eta$ $= (x^{-1}.m^q, e_1) \bmod \eta$ |
| <i>Compute $(x.m^r)^{e_2} \bmod \eta$ and delivers it to client B.</i> | <i>Compute $(x^{-1}.m^q)^{e_1} \bmod \eta$ and delivers it to client A.</i> |
| <i>Decrypt the newly received key-part $(x^{-1}.m^q)^{e_1.d_1}$</i> <i>Compute $(x.m^r).(x^{-1}.m^q) \bmod N$</i> <i>Session key m^{r+q} is established</i> | <i>Decrypt the newly received key-part $(x.m^r)^{e_2.d_2}$</i> <i>Compute $(x^{-1}.m^q).(x.m^r) \bmod N$</i> <i>Session key m^{q+r} is established</i> |
| Table 12 : Mathematical computations required at server/clients end | |

Given that our protocol makes use of RSA encryption/decryption, it seems computationally expensive but the actual computation required at both ends is less than that of actual length of the session key generated. Let's say the server generates two key-parts (ephemeral keys) $x.m^r$ and $x^{-1}.m^q$ which are 2^5 bits (32 bits). The server performs RSA encryption of these two 32 bits integers. The receiving clients first decrypt the protocol message (encrypted key-part and public key of the other client) to get their 32 bit key-part and then encrypt it again with the public key of the other client. The length of the generated session key is $\approx 2^6$ (64) bits, i.e., the computational efforts required to establish 2^x bits of session key is almost (\approx) 2^{x-1} bits of computations required at the server and clients' end.

Note: Clients-end computations are more than that of the server-end, i.e., there are two encryptions at the server-end while there are two decryptions, one encryption and one modular multiplication at the clients-end.

5.7 Deployment in a Closed Environment

The most reliable and well known mode of authenticating clients in a distributed environment is the use of certificates. In a closed environment, i.e., when number of clients in the system at any instance could be determined, the trusted server identifies each client in the network, e.g., in a wireless network a base station typically identifies each specific mobile user who is tuned into the system. In such cases, since clients can be easily identified by the TS (or the base station) password-based authentication or a self-signed certificate may suffice the entire authentication process.

One of the earliest AK-protocols for use in a mobile environment called TMN [Tatebayashi|90] makes use of the server to mediate the key establishment between two entities. The authors proposed the use of small RSA public key exponents taking into account the limitations of the mobile device's computational ability.

On the contrary, Hansen et al. [Hansen|10] proposed efficient fast RSA variants for modern mobile phones, which can be used to achieve fast computations on the server and client ends.

Since, the traditional public-key cryptosystems restricts their application in a resource constrained environments, Wang et al. [Wang-Wei|09] came up with a fast public-key cryptosystem that uses the Chinese Remainder Theorem [CRT]. It speeds up the encryption by carrying out several modular-multiplication operations, and the decryption only needs a modular multiplication and a low-dimensional matrix-vector multiplication. The attacker has to solve the integer factorization problem and the Simultaneous Diophantine Approximation Problem [Lagarias|82] to recover the secret key from the public key.

Morchon et al. [Morchon|09] presented a novel scheme which allows fast, resource-friendly and distributed key agreement. According to their scheme, verification of the client's information makes use of the efficiency of symmetric-key cryptography and the functionality of public-key certificates. The underlying concept allows the base station of a sensor network to sign node identification and configuration information such as routing addresses or access control roles by means of the polynomial shares distributed to nodes creating a lightweight digital certificate for each node. The system operates in a fully stand-alone and distributed way, being able to perform a combined key agreement and lightweight digital certificate verification handshake within a few milliseconds with very low memory requirements.

With the advent of highly efficient mobile phones it is now possible to have algorithms like RSA work on them [Boyd|00]. For the AK-protocol in a mobile environment we suggest that the participating clients authenticate themselves when they select the feature of secured communication called *secured services*. Since, in a mobile environment to-and-fro transmissions are more time and power consuming, it would be wise to reduce the protocol messages required to authenticate clients each time they opt for secured communications. We suggest that mobiles with higher computational capabilities may use RSA-1024 bits or more since RSA-768 bits has already been factored [Kleijnung|10], while less-valued real-time secured voice communications may choose smaller exponents, *an example is shown in Section 6.5.3*. Another option is the use of fast RSA variants as proposed in [Hansen|10] and [Wang-Wei|09].

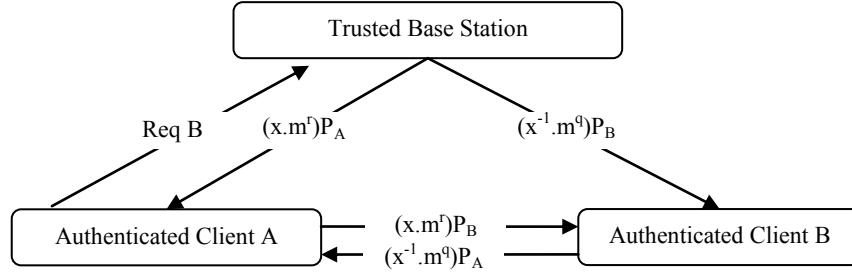


Figure 5 : The Protocol in Mobile Environment

As shown in the Figure 5, when the clients turn on the secured services in their mobile devices they are first authenticated by the trusted server or the base station. After which a client may request the TS to initiate a communication with the other clients that have enabled secured services. When the TS receive a request from client A, it distributes the ephemeral keys to both participating clients A and B. They then exchange their key-parts to complete the key-establishment process.

5.8 Properties of the Protocol

Wilson and Menezes [Wilson-Menezes|97, Wilson-Menezes|98] have defined a number of desirable security attributes which are normally used to analyze any key agreement protocols. In this section we will discuss only the relevant properties from the ones listed in Section 3.11. There are some more properties, e.g., key distribution and desired properties in a 3PAKE which are not listed in the above reference but we would consider them to analyse the protocol in all aspects.

5.8.1 Perfect Forward Secrecy

A protocol is said to have the property of perfect forward secrecy if the compromise of long-term private key of a client does not compromise past session keys ($m^{(r_{old} + q_{old})}$). Depending upon the pre-defined architecture of the AK-protocol there can be two different cases:

- a. When the trusted server does not store the protocol messages in the database, e.g., in a mobile environment trusted server might not want to save all the session keys that were used in the past to encrypt real-time voice communications. In this situation there is no possibility of compromise of old session keys since neither trusted server nor any of the clients has saved the past session keys.
- b. When the trusted server stores the protocol messages of the previous sessions, i.e., $(x_{old}.m^r_{old}, P_y)P_x$ and $(x^{-1}_{old}.m^q_{old}, P_x)P_y$ in a secured database. An adversary in pursuit of retrieving the old session keys has to first breach into the secured database, which is accessible only to the trusted server, to get these two old encrypted session key-parts (because as per the architecture of protocol, clients do not save old session keys but they save the transaction ID for future reference). The database stores the two (distinct) key-parts which are encrypted by two different public-keys. If the adversary somehow acquires these two encrypted key-parts and has the private key of one of the clients then he can decrypt only one of the key-parts. The bit complexity is $O((1 + (\log_m N)^2) \log_m N)$ operations to recover the complete session key. This means that an adversary has a chance of recovering the old session keys if he has the capability to break into the secured database of the trusted

server, and had acquired the long-term private keys of both the participating clients.

5.8.2 Known Session-key Security

A protocol is said to be resilient to known session-key attack if the compromise of past session keys ($m^{r_{old} + q_{old}}$), does not allow (i) a passive adversary to compromise future session keys, and (ii) impersonation by an active adversary in the future. In the AK-protocol, the trusted server generates the two distinct ephemeral keys or key-parts that make use of three random parameters: exponents' r and q , and a pair of multiplicative inverses. The permutations of these three parameters are different for every session. Furthermore, they do not depend upon (a) any inputs from the participating clients, that is, these random parameters, or (b) the ephemeral keys cannot be set to pre-determined values. Acquiring and analyzing an old session key would not give leverage to an adversary to predict the random parameters $r, q, (x, x^{-1})$, the trusted server would generate next. A scenario when adversary E steals client A's certificate to impersonate him is shown in Section 5.9.3 Case (e).

5.8.3 Unknown Key-Share Resilience

In an unknown key-share attack, an adversary convinces or tricks a client to share his long-term private key or session key or ephemeral keys. The entire protocol rests on the keeping the long-term private key secret. The architecture of the protocol has no provision for the long-term private keys to be shared with any arbitrators including trusted server; the clients can only share this critical information with their CA via a secondary channel. However, let's consider situations where an adversary can trick any/both

participating clients to share their protocol messages (contains encrypted ephemeral key). The AK-protocol acknowledges public-key encryption of the protocol messages communicated between the server-to-clients and client-to-client. If an adversary has somehow acquired all protocol messages, it would still require a bit complexity of $O(\eta \cdot (2 \cdot (\log_m N)^3))$ to recover the session key as shown in Section 5.9.2, equation (c.2).

5.8.4 Key-Control

As discussed in Section 5.8.2, the trusted server employs three randomly generated parameters for ephemeral key generation and any participants (or an adversary) cannot compel the session key to a preselected value or predict the value of the session key. The possibility of pre-computation to reduce on-line computational complexity to unravel a session key depends on the random number generator⁹ used by the trusted server. However, the control of generation of ephemeral keys exclusively lies with the trusted server and the session key is generated at both ends only after the participating clients have exchanged their ephemeral keys. This implies that the key establishment process depends on the trusted server and both participating clients.

5.8.5 Desired properties in a 3PAKE

Chen et al. [Chen-Lee08] presented three general issues which are discussed below, with any three-party authenticated key establishment protocol (3PAKE) and their solutions are:

⁹ A Random Number Generator is a program that produces a number each time it is invoked. More exactly, it is unpredictable and has the same chance of being selected as any other number among those the generator can produce.

(1) *To reduce the latency a protocol should provide fewer rounds*, in our protocol key establishment is accomplished in two rounds (as the two messages from the server-to-clients are delivered in parallel and there is simultaneous exchange of messages between the clients). However, the server is set-free after the first round.

(2) *Saving the keying material would lure an attacker to attack the trusted server in desperation*, which the authors suggested to remove. In our protocol, trusted server stores the encrypted key-parts if the system calls for non-repudiation. Besides, the two key-parts are individually encrypted with the public key of both the participating clients.

(3) *Reduced computation to avoid bottleneck at the server*, as per our protocol for key establishment the trusted server performs two operations: (a) certificate verification to authenticate clients, and (b) RSA encryption of two ephemeral keys for authenticated participating clients. Here, the certificate verification refers to the validity of the certificate. RSA encryption however is an expensive operation but considering the server is not dependent on any parameter from either client to generate these two ephemeral keys avoids the wait-state (is a state when server is expecting some operations/process/data from client-end). Additionally, after delivering the (encrypted) ephemeral keys to the clients, the trusted server can disconnect itself from them, i.e., the server does not stay connected with the clients after it has delivered the ephemeral keys to the participating clients. This mechanism prevents an adversary to open a number of connection requests to the server while keeping it in wait-state. But still the trusted server is a single point of contact which could lead to bottleneck. An effective way to reduce this, but again not eliminate the possibility of

bottleneck is to have multiple trusted servers that provide service to the clients.

5.8.6 Key freshness and Key compromise

In the AK-protocol, the element of *freshness* is added by the parameters: pair of multiplicative inverse (x, x^{-1}) chosen randomly from the *poi* (Pool of Multiplicative Inverses), and exponents r and q . *If an adversary cannot distinguish a session key from a randomly selected string with non-negligible probability, then the session key is viewed as secure and suitable for use in bulk encryption* [Stebila-Ustaoglu09].

5.8.6.1 Ephemeral-key Compromise

Compromise of key-parts $(x.m^r)^{e^A}$ and $(x^{-1}.m^q)^{e^A}$ of client A would require a bit complexity of $O(\eta ((1 + (\log_m N)^2) \log_m N))$, as shown in Section 5.9.2, equation (b.3), for an adversary to recover the session key. The adversary has to acquire all the protocol messages between server-to-clients and client-to-client to comprehend the above mentioned key-parts (i.e., the two key-parts encrypted with the same public key e^A), as discussed in Section 5.9.3, Case (e).

5.8.6.2 Long-term Key compromise

When a long term private key of any participating client is compromised, and the adversary who now possesses the private key of the one of the participating client also captures the protocol messages transmitted between the trusted server-to-clients *and* client-to-client, then he can compute the modular multiplication over N (provided he has the knowledge of modulo N) to uncover the session key. However, according to our architecture either

the clients or CA generate the key-pair (and it is exchanged over a secondary channel), so we assume the liability of the private key either lies with the issuer and/or, the client. The clients can generate a new pair of long-term keys or send a request to CA to do so, in the event of realization of key compromise.

5.8.7 Bandwidth required

The server-to-client protocol message has two components separately encrypted (over modular operation η): (i) the public key of the other client, and (ii) the key-part $x.m^r$ (or $x^{-1}.m^q$). The client-to-client protocol message, on the other hand, consists of a single component, i.e., encrypted key-parts (over modular operation η). Therefore, the maximum number of bits that needs to be transferred from server-to-client is $\approx \text{length}(2\eta)$, and client-to-client is $\approx \text{length}(\eta)$.

5.8.8 Scalability

In a traditional system where each client stores the public keys/certificates of all the other clients it wishes to communicate, for example, if the system has n number of users at an instance then each client needs to save $n-1$ public-keys/certificates. It is imperative the clients have a correct list of client and their corresponding public-key/certificate, therefore in case of a new user joining in the system each user is required to add the public-key/certificate of this new client to their list. While the same effort applies to when an existing user leaves the system, the list is required to be updated to delete the public-key/certificate of the client leaving the system. However, this process thwarts the scalability of the system which is best

dealt with a centralized separate entity certification authority for generation of public-key/certificates.

According to AK-protocol, either a client or a CA is responsible for long-term key generation while trusted server is independent of clients joining in or leaving the system. This gives liberty to the entire system to expand and contract independent of the trusted server. This means that any number of clients can join or withdraw from the system, without actually affecting the existing structure or functionalities of the protocol which makes the system highly scalable and suitable for large distributed environment.

5.8.9 Key distribution

In a typical system, a server which is responsible for generating the key pairs also takes charge of authenticating the communicating clients. However, in the AK-protocol either the client or a CA is responsible to generate the key pairs after which CA securely publishes the issued public key certificates to the trusted server, who eventually authenticates the clients. As discussed in Section 5.4, we have two separate entities one that issues the certificates (*the issuer or CA*) and another that authenticates the clients (*the trusted server*). The trusted server is not involved in the process of long-term key generation and distribution.

5.8.10 Centralized Directory Service

Revoked certificates are published in a list called *Certificate Directory* (see Section 4.5 for more information) which is available publicly. In the systems where a client authenticates the public key certificate of the other client they have to either constantly update their cache for the latest published certificate directory, or add extra overhead to check it online in an

OCSF (see Section 4.5 for more information) for each transaction. While in the systems where a trusted server is responsible for authenticating the clients, this directory service is only required at the server's end. This reduces the extra overhead of updating the secured directory service and making it available for individual clients.

5.8.11 Non-Repudiation

Optionally, trusted server stores the delivered encrypted protocol messages $((x.m^r, e_2)^{e_1} \bmod \eta \text{ and } (x^{-1}.m^q, e_1)^{e_2} \bmod \eta)$ in the secured database indexed with a unique transaction ID, for each communication. The clients however, do not save the session keys after the session is completed, but they do save the transaction ID for future reference. Hence, in the occasion of repudiation from any party, the trusted server can retrieve the stored protocol messages for that particular transaction (trusted server has no knowledge of the client's private keys) and can instigate either the participant clients or the CA to decrypt the encrypted key-parts (since, the CA also saves the public and private keys of the clients in its database for the purpose of key escrow). These key-parts when multiplied over modular operation N produce the session key which was used for that transaction.

By doing this, the system not only encompasses the non-repudiation factor but also impedes impersonation attacks. Let's take an example when an adversary E steals the certificate of a client A but could not acquire his private key. E then impersonates as A , denoted by $A(E)$, to initiate a communication with B .

trusted server after verifying the $A(E)$'s certificate delivers two key-parts to $A(E)$ and B and also records them in its database, indexed with a unique transaction ID. However, $A(E)$ could not establish a connection with B

because it could not decrypt the key-part it was supposed to exchange with B. The session key is not established unless the clients have successfully exchanged their key-parts. So, the session is said to be incomplete, i.e., A(E) and B could not establish a session key.

Regardless, trusted server has already stored this attempted transaction in its database, which may assist the suspected future cyber crime efforts. But, we must also keep this in mind that compromise of the client's private-key would put the system at high risk.

5.8.12 Message Flow

The types of protocol messages that are exchanged between a server and a client are mentioned in the Table 14. This message flow gives an objective overview of the functions required to create that will facilitate while the implementation of the protocol.

| | Sends | Receives |
|--|---|--|
| Server | Req(CertX), Message(X) Cert(trusted server) [optionally] | Req(X) Cert(X) |
| Client | Req(X) Cert(X) Message(X) Req(CertTS) [optionally] | Req(CertX), Message(X) Cert(TS) [optionally] |
| Table 13 : Types of Protocol Messages | | |

5.8.13 Key Escrow

A key escrow or key recovery scheme allows law enforcement authorities, or other authorized persons to decrypt cipher-text with the help of key escrow/recovery information supplied by one or more TTPs under special prescribed conditions.

As per the architecture of the protocol, either a client or a CA generates the long term key pairs and is definitely secured within the domain of the CA for the benefit of key escrow. Any exchange of private-key between a CA and the clients is done securely via a secondary channel.

When a session key needs to be recovered the CA may request the trusted server to retrieve the encrypted key-parts indexed by transaction ID from its database. Thereafter, these encrypted key-parts can be decrypted by the CA who possesses the private keys of the participating clients to recover the key-parts after which a session key can be computed (by applying modular multiplication of the key-parts).

5.9 Security Analysis of the protocol

This section presents different scenarios when an adversary may possibly interfere with the protocol messages. In all the cases below E is considered as an adversary.

5.9.1 Replay Attack

A replay attack is when an adversary tries to capture and record the protocol messages of a session from the wire with the intention of replaying it later to the benign clients, or to unmask the session key from the captured

messages. As shown in the Figure 6, there can be three ways of capturing the protocol messages while they are communicated. Certainly, the adversary E needs the private keys of clients A and B to actually uncover the session key, because all the server-to-client and client-to-client protocol messages are encrypted by the respective recipient's public keys.

But, let's assume that E just records these messages to replay it later to the clients A and B. The protocol messages do not include a timestamp and the element of *freshness* is added by the random integers (exponents' r and q) and a pair of multiplicative inverses.

We assume for the moment that the trusted server cannot be impersonated; we shall revisit this assumption in Section 5.9.4.

Note: Protocol messages delivered to the clients without any authentication are detected as false messages.

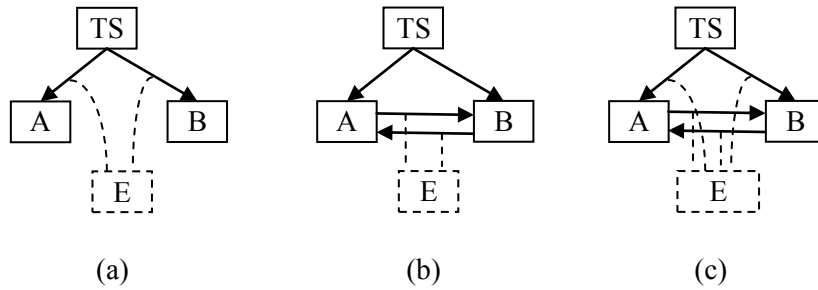


Figure 6: Adversary capturing the protocol messages.

Case (a): An adversary E capable of capturing the messages $(x_{\text{old}.m^{r1}}, P_B)P_A$ and $(x_{\text{old}.m^{q1}}^{-1}, P_A)P_B$, delivered by trusted server to the clients A and B may record these messages to replay later (as there is no timestamp included in them). E then intently listens to the communications between A and B

and delivers these recorded messages. For the current session, clients A and B may have also received a set of fresh messages $(x.m^r, P_B)P_A$ and $(x^{-1}.m^q, P_A)P_B$ from the trusted server. Now, the clients have two sets of messages, either they discard all the messages and start again, anyhow the source of the messages can always be identified so the messages from the trusted server could be retained.

However, E still has no knowledge of the old session key but is able to hinder the session key establishment in progress between two benign clients. A solution to this problem is to append hash of the key-parts signed by the trusted server, along with the protocol messages to ensure data origin authentication, discussed more in Case (f).

Case (b): When E can capture and record the protocol messages that are communicated between the clients, $(x.m^r)P_B$ and $(x^{-1}.m^q)P_A$. However, the clients detect it as *false messages* as there was authentication done by the trusted server prior to the delivery of these messages.

Case (c): Where E could capture all four messages $(x.m^r, P_B)P_A$, $(x^{-1}.m^q, P_A)P_B$, $(x.m^r)P_B$ and $(x^{-1}.m^q)P_A$ with the intent to replay it to the clients at a later time. When E replays these messages to A and B, either of the two above mentioned cases will arise.

5.9.2 Bit Complexity

The bit complexity of the key-part $x.m^r \pmod{N}$ is:

$$\begin{aligned} O(\log x + \log m^r) &= O(\log_m N + (\log_m N)^3) \\ &= O((1 + (\log_m N)^2) \log_m N) \quad \dots\dots(b.1) \end{aligned}$$

And the bit complexity of the session key (m^{r+q}) would be

$$O((\log_m N)^3 + (\log_m N)^3) = O(2 (\log_m N)^3) \dots\dots\dots (b.2)$$

The bit complexity of the protocol message $(x.m^r \bmod N)^{e_A} \bmod \eta$

$$= O(e_A \cdot (\log x + \log m^r)) = O(\eta \cdot (\log_m N + (\log_m N)^3))$$

$$= O(\eta ((1 + (\log_m N)^2) \log_m N)) \dots\dots\dots (b.3)$$

Where m is the generator of the cyclic group Z_N^* and m' is the generator of the cyclic group Z_η^* .

Note: The bit complexity of the modular operation $\log(a^k \bmod n)$, $k < n$ is $O((\log n)^3)$ as per [Menezes|96], Chapter 2, Page 72, Table 2.5.

Case (d): When an adversary E possesses all the protocol messages communicated between trusted server and clients, i.e., $(x.m^r, P_B)P_A$, $(x^{-1}.m^q, P_A)P_B$; $(x.m^r)P_B$ and $(x^{-1}.m^q)P_A$ with an intent to either cryptanalyze or recover the session key from the captured protocol messages. Adversary E , possesses the protocol messages: $(x.m^r, e_B)^{e_A}$, $(x^{-1}.m^q, e_A)^{e_B}$, $(x.m^r)^{e_B}$, $(x^{-1}.m^q)^{e_A}$, mathematically. To calculate the bit complexity let's keep aside the public key component (e_A , and e_B) from the messages so E can perform a modular multiplication of $(x.m^r)^{e_A}$ and $(x^{-1}.m^q)^{e_A}$.

$$= (x.m^r \cdot x^{-1}.m^q)^{e_A} \bmod N = (m^{r+q})^{e_A} \bmod N \dots\dots\dots (c.1)$$

$$\text{The bit complexity} = O(e_A \cdot (\log(m^r) + \log(m^q)))$$

$$= O(\eta \cdot ((\log_m N)^3 + (\log_m N)^3))$$

$$= O(\eta \cdot (2 \cdot (\log_m N)^3))$$

$$= O(2\eta (\log_m N)^3) \dots\dots\dots (c.2)$$

The adversary E will have to deal with RSA factorization and Integer factorization of two different modulo operations to recover the session key as discussed in Section 5.9.6.

5.9.3 Impersonation Attack

Case (e): When E steals A's certificate to impersonate him.

Assuming that E has not acquired A's private key but he has acquired A's certificate which he can present to trusted server thus pretending himself as A (denoted by A(E)). When A(E) initiates the communication with trusted server to establish a session key with B by sending its certificate. Upon receiving A's certificate from A(E), trusted server would verify the certificate and would send two messages to A(E) and B. After B had send his key-part, A(E) will have $(x.m^r)^{e_A}$ and $(x^{-1}.m^q)^{e_A}$. The bit complexity to get the session key would be same as Section 5.9.2, equation (c.2). This implies that the adversary E would require the private key of the client A to fully impersonate him.

5.9.4 When an Adversary can Craft Protocol Messages

We have been assuming so far that the trusted server cannot be impersonated. But let's say some highly sophisticated, diligent adversary has gained knowledge of the system and has the capability to act as the trusted server, i.e., he can craft the protocol messages. This situation can arise when the system makes use of one-way authentication, i.e., only client authenticates itself to the server. However, this scenario will not appear in

case of two-way authentication, i.e., when trusted server also presents its certificate to the participating clients for validation and verification.

Case (f): Subsequently, E who is now acting as trusted server (denoted by TS(E)) can now create two messages of his choice say, $(y.m^s)$ and $(y^{-1}.m^t)$. When A wants to communicate with B, TS(E) delivers these two “crafted” messages to A and B. After A and B exchange their key-parts they compute the session key, but this session key is already known to TS(E). Therefore, he can now decrypt all the communication that was supposed to be secret.

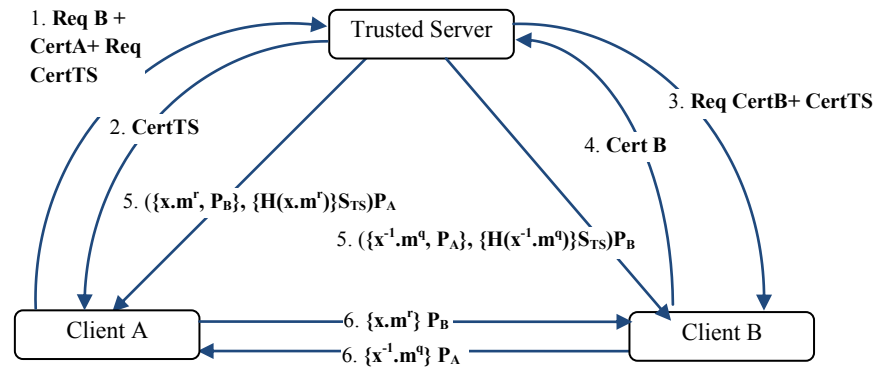


Figure 7 : Signed Protocol messages with two-way authentication

Subsequently, when the system makes use of one-way authentication, the existing protocol might need an enhanced version of the protocol message to achieve data-origin authentication. To achieve this a hash component is added while delivering protocol messages $(x.m^r)P_A$ and $(x^{-1}.m^q)P_B$, from the trusted server to the clients. As shown in Figure 7, the hash of the key-part is signed by TS which is appended to the original protocol messages, which looks like $\{(x.m^r), (\{H(x.m^r)\}S_{TS})\}P_A$ and $\{(x^{-1}.m^q), (\{H(x^{-1}.m^q)\}S_{TS})\}P_B$.

When the participating clients receive this signed hash they can verify whether the message is coming from their *trusted* server. This gives leverage to the system that even after trusted server is disconnected from the clients, the clients have a means to verify the source of the message, i.e., data origin authentication is achieved eventually mitigates the adversary to impersonate trusted server.

The outcome of this added mechanism is one more computation at the server end to compute the hash and sign it and its verification at the client's end. The bandwidth of server-to-clients protocol messages increases to $\approx 3\eta$ ($2\eta + \text{Length (Signed Hash)}$) while the bandwidth of the client-to-client messages remains same, i.e., η .

5.9.5 Distributed Denial of Service (DDoS) Attacks

Key establishment protocols are applications that are particularly vulnerable to DDoS attack, as they are required to perform computationally expensive cryptographic operations. In a three party key establishment protocol, typically a trusted server is present to authenticate the participating clients, and generate the ephemeral keys or key-parts that will subsequently be used to compute the session key to secure the communications between the clients. The primary goal of DDoS resistance is to ensure that the attackers are not able to prevent the legitimate clients from deriving cryptographic keys without expending their resources beyond a determined threshold.

Case (g): When a key establishment protocol exhibits certain properties as proposed by the authors Stebila and Ustaoglu [Stebila-Ustaoglu09], it is said to be DDoS resilient. The authors suggested challenge and response

based client puzzles to offer DDoS resilience. J. Smith and et al. [Smith-Tritilanunt|07] suggested three strategies for DDoS resilience: counterbalancing computational expenditure, counterbalancing memory expenditure and gradual authentication. As stated by the authors that the informal criteria for a DDoS resilient key establishment protocols are:

- (a) Depending upon the kind of environment, the server should avoid performing unnecessary expensive operations. For example, in a mobile environment to-and-fro transmission is more time and power consuming. The authors have identified three main classes of DDoS attacks:
 - i. *Memory DDoS attacks*, where the server is forced to perform slow and expensive memory read/write operations, or make use of a large amount of memory, to eventually degrade the performance of the server. As per the architecture of the AK-protocol the trusted server is stateless as discussed in Section 5.5 phase 2, which makes the protocol resilient to memory attacks. However, in case of benefiting the system with non-repudiation capabilities, trusted server does store the encrypted key-parts in its secured database. But, this write operation is performed after the trusted server has delivered protocol messages to the participating clients so we can say that it does not suspend the process of key establishment. However, in practice the secured database resides within the trusted server and with efficient memory management systems offered by various operating systems makes it of diminutive concern.
 - ii. *Computational DDoS attacks*, where the server is forced to perform operations involving significant amount of

computational time (such as exponentiation, elliptic curve point multiplication, or even a large number of simpler operations such as hash function or MAC evaluations). We demonstrated in Section 5.6 that the number of computations required for generating 2^x bits of session key is approximately (\approx) 2^{x-1} bits of computations at server and client sides. This implies that the strength of the generated session key is almost half of the computational efforts required at both ends. Additionally, the participating clients are first authenticated before the trusted server would undergo expensive RSA operations involved in the process of key generation. An adversary will have to successfully accomplish authentication mechanism to further engage trusted server in such expensive operations.

- iii. *Transmission DDoS attacks*, where the server is forced to expend its entire available resources for to-and-fro communications. Although, this evaluation criterion is significant to the wireless environment because the cost, effort and bandwidth of communicating protocol messages is crucial. In a wired connection the number of rounds of protocol messages communicated between participants affects bandwidth. As suggested in Section 5.7, our protocol is different in case of mobile users who are already authenticated by the trusted server (or the base station) upon tuning in the “secured services”. Therefore, the number of transmissions required to establish a session key is reduced to only two rounds of messages (instead of four), i.e., server-to-

clients and client-to-client making the cost of communicating protocol message to half.

- (b) A client investing significant resources to prove its legitimate intentions should not benefit others to establish a session, i.e., no one should be able to steal a client's work and use it in another pre-session. We have discussed this possibility by taking various cases as illustrated in Case (a-f). From these cases we understand that no one else would be able to capture or intercept the protocol messages from a legitimate client and use it for their own session establishment.
- (c) To possibly flood the server an adversary must use a significant amount of resources. As per the AK-protocol, if an adversary wishes to open sufficiently many valid connection requests with an intention to flood the server, he may require tricking a lot of legitimate clients to reveal their private keys or copy their legitimate certificates. However, the clients are well informed of the fact that private keys are never shared online with any mediator (not even trusted server), except the authorized CA via a secondary channel. So, there are two possibilities for flooding the server, when somehow the adversary E manages to steal a number of legitimate certificates:
 - i. To launch *Reflection attack* where the target is not actually the trusted server but another client, E creates a number of protocol messages, for example, (ReqB, CertE1), (ReqB, CertE2),....., (ReqB, CertEn) and sends it to the trusted

server to connect to a single client. However, this can easily be detected at the application layer.

- ii. To flood the trusted server, E creates a number of protocol messages requesting different clients, e.g., (ReqB, CertE1), (ReqC, CertE2),....., (ReqZ, CertEn), this means that the adversary is engaging the server to perform a number of certificate verification operations.

These two attacks can be mitigated if the important credentials like certificates are password protected, i.e., a client has to enter his secret passphrase every time he wishes to access his certificate. This way just stealing the certificates would not let an adversary attempt reflection or flooding attacks.

There is different kind of DDoS attack where an adversary leaves a large number of half open TCP connections, the clients maintaining the states of these half open connections may lead itself to DDoS attack. In our protocol the trusted server is stateless because when it receives the initial request it generate and delivers the ephemeral key-parts to the clients after which it may detaches itself from the clients, which makes it resilient to this kind of attack.

- (d) Balancing the load between server and client at the event of DDoS attack. This kind of load balancing mechanism is best handled at the application level instead of at the protocol level [Jerschow|09].
- (e) Our protocol does not follow the strategy of gradual authentication¹⁰ since the participating clients are authenticated before any exchange

¹⁰ Gradual authentication provides a mechanism for weakly authenticating an initiator, prior to performing stronger and more expensive cryptographic authentication.

of the protocol messages. As suggested in [Smith-Tritilanunt07], gradual authentication assists in verifying the computational and memory commitments of the initiating client and associates the source of the protocol messages while our protocol caters to these benefits through other ways as discussed in Case (a) and Case (f).

5.9.6 Cryptanalysis

There are four different layers of mathematical complexity to deal with, when a cryptanalyst wishes to crack the AK-protocol. The overall bit complexity involved while generating the session key m^{r+q} , as shown in Section 5.9.2, equation (b.3). To acquire the session key m^{r+q} , a cryptanalyst has to discover the values of exponents' r and q , used while creating the key-parts. Let's have a look at the how exponents' r and q can be recovered from messages by mathematically reverse engineering the protocol messages:

$$((x.m^r \bmod N) P_A) \bmod \eta \text{ and } ((x^{-1}.m^q \bmod N) P_B) \bmod \eta$$

1. RSA Factorization, since the protocol messages are sent encrypted with RSA algorithm over modulo η (see Section 5.9.7 and Definition 17 for more information).
2. Integer factorization of modulo N ($< \eta$) (see Section 5.9.7 and Definition 16 for more information). Since key-part $(x.m^r)$ is first operated over modulo N and then modulo η for encryption.
3. Pair of multiplicative inverse (x, x^{-1}) adds complexity equivalent to length of N [Menezes|96].

4. Finally, to get both exponents' r and q a cryptanalyst will have to solve the problem of discrete logarithm (DLP) (Section 2.6, Definition 1).

These four layers of mathematical complexity add to the strength of the generated session key and make the overall system robust.

5.9.7 Integer Factorization and RSA Problem

A computational problem is said to be easy or tractable if it can be solved in expected polynomial time, at least for a non-negligible fraction of all possible inputs. The security of RSA encryption and decryption depends upon the intractability of the integer factorization problem [Brent|00, Brown|06, Joux|07 and Aggarwal|09].

Integer factorization or prime factorization is breaking down a composite number into smaller non-trivial divisors, which when multiplied together equals the original integer. Every positive integer has a unique prime factorization. However, no efficient integer factorization algorithm is publicly known for large numbers. The presumed difficulty of this problem is at the heart of RSA algorithm.

However, there has been a constant diligent effort on breaking the RSA as described in these research papers [Blakey|09, Heninger|09, Finke|09, Sarkar|09, Nitaj|09, Sun-Wu-Chen|07 and Jochemsz|07].

Kleinjung et al. [Kleinjung|10] have successfully factored RSA-768 bits, they stated that the computations required to factorize 768-bit RSA were more than 10^{20} operations which would be equivalent to almost 2000 years of computing on a single core 2.2GHz AMD Opteron on the order of 2^{67} instructions were carried out. They concluded that "Factoring a 1024-bit

RSA modulo would be about a thousand times harder, and a 768-bit RSA modulo is several thousand times harder to factor than a 512-bit one.” Another conclusion from their work is that “we can quite confidently say that if we restrict ourselves to an open community, academic effort as ours and unless something dramatic happens in factoring, we will not be able to factor a 1024-bit RSA modulo within the next five years”. J Milan [Milan|10] has presented the current practicality of different factoring methods with an emphasis on small to medium-sized composites (50 to 200 bits) with no small factors.

As stated by Shamir and Tromer, “using a hypothetical device (and ignoring the initial R&D costs), it appears possible to break a 1024-bit RSA key in one year using a device whose cost is about \$10M” [Shamir-Tromer|03]. Nevertheless, all the computations would be futile if the value of the information is less than this value, or time taken to crack the information is more than the value of the information.

Chapter 6

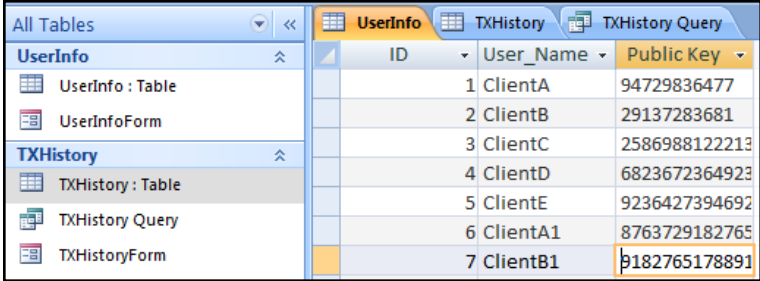
Implementations

6.1 Chapter Introduction

This chapter describes the implementation aspects of the AK-protocol in different practical situations. It is divided into four major sections: First, we discuss the Java implementation of the AK-protocol on TCP layer. Second, we discuss the possibility when the AK-protocol would be integrated with the existing 3-D secure protocol of MasterCard and Visa. Third, we suggest the integration of the AK-protocol for wireless environment (a) with the mobile payment systems, and (b) for securing real-time mobile communication between two users, where generation of session key is accomplished by the AK-protocol after which a fast stream cipher like RC4 is used for encryption and decryption of real-time communication. Fourth, we present three examples that illustrate the cryptographic mathematics involved in the AK-protocol.

6.2 Implementation of the AK-protocol

We created a proof-of-concept implementation of the AK-protocol on TCP layer using Java for closed environment. This Java project is a multi-threaded client-server based architecture where the server can accept multiple clients at any instance of time. The prototype consists of the server and client modules: The server module listens to the TCP connection for connecting clients, while each client module is provided with two different ports, one to listen and another to connect, as the client has to first connect to the server for authentication thereafter to other participating client for exchange of the protocol messages. At the time of initialization we created a secured database, accessible only to the trusted server that stores the certificates of the clients registered in the system, as shown in Figure 8. The server can connect to this local MS Access database to verify the certificates presented by the clients.



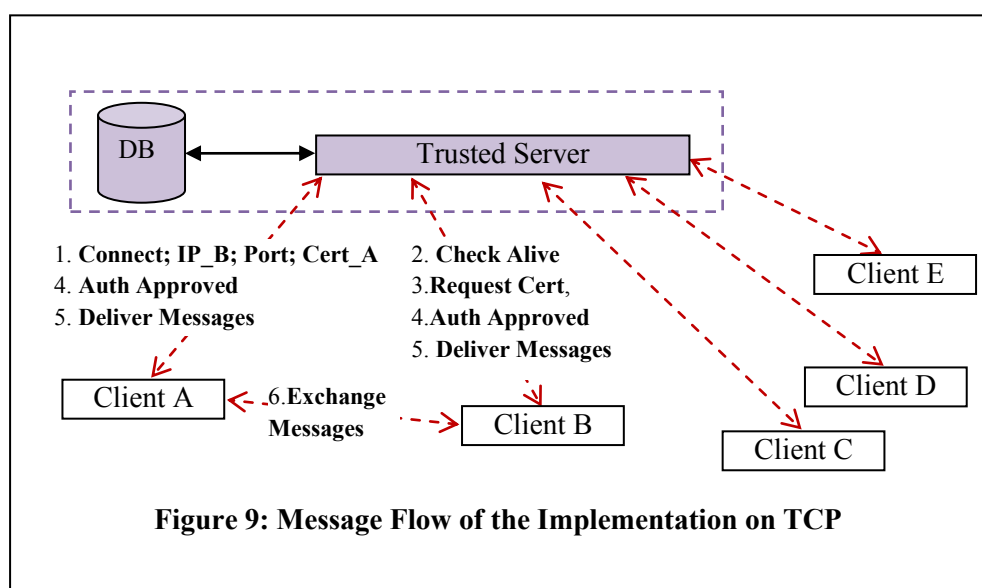
| ID | User_Name | Public Key |
|----|-----------|---------------|
| 1 | ClientA | 94729836477 |
| 2 | ClientB | 29137283681 |
| 3 | ClientC | 2586988122213 |
| 4 | ClientD | 6823672364923 |
| 5 | ClientE | 9236427394692 |
| 6 | ClientA1 | 8763729182765 |
| 7 | ClientB1 | 9182765178891 |

Figure 8: DB of Clients' information

The dotted line between the server and clients, as shown in Figure 9 indicates that they are connected. After the server module is started, the clients can connect to its listening port. When client A wants to talk securely to client B, it sends request to the server with its certificate, IP and port of

client B (1). In this prototype we are assuming the other clients know each other's IP and port information. In real situation the trusted server will have to resolve the IP and port of the requested clients.

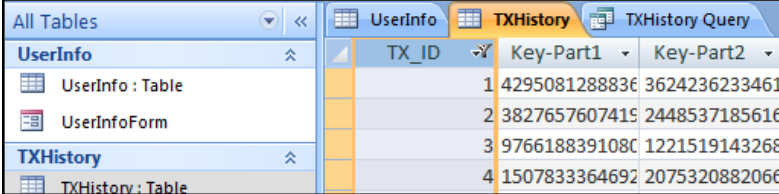
The server first checks whether client B is alive or connected (2), then it requests client B for its certificate (3). After client B presents its certificate the server verifies certificates of both clients, if they are identified by the trusted server, it notifies approved authentication (Auth_approved) to both of them (4). The server now generates the key-parts, appends the public key of the other client and encrypts it with the public keys of respective clients, and delivers it (5). The server stores the encrypted key parts indexed by the transaction number in a secured database as shown in Figure 10. The clients can now connect with each other to finish the key exchange and compute the session key (6).



The generated session key is used as a key to any symmetric key (block cipher or stream cipher) or asymmetric key algorithms to secure further communications. We recommend use of any block cipher algorithm (e.g.,

AES, 3DES) for wired connections and stream cipher algorithm (e.g., RC4) for wireless communications which is discussed more in Section 6.4.3.

If we decide on AES as an algorithm for encryption and decryption then the required key length is 128, 192 or 256 bits. For example, to generate 128 bits of session key (i.e., m^{r+q}) we must satisfy two conditions: (a) Modulo $N > 128$ bits and (b) $m^r, m^q \approx 64$ bits. In general, if the desired length of session key is x bits, then $N > x$ bits and parameters $m^r, m^q \approx (x/2)$ bits. A padding scheme could be used for session keys which are less than the desired key length.



| TX_ID | Key-Part1 | Key-Part2 |
|-------|---------------|---------------|
| 1 | 4295081288836 | 3624236233461 |
| 2 | 3827657607419 | 2448537185616 |
| 3 | 9766188391080 | 1221519143268 |
| 4 | 1507833364692 | 2075320882066 |

Figure 10: DB of the transactions

The snapshots shown in Figures 11, 12 and 13 are from the Java implementation of the AK-protocol. For our convenience we are running the server and the clients in localhost, however they worked in different machines while conducting our experiments. For the benefit of understanding the flow of protocol messages we are displaying the values on the screen. These values are presented in the examples as specified in Section 6.5, other snapshots of the examples mentioned in the same section can be found in Appendix B. We have intentionally chosen small parameters to have all the values displayed in a single screenshot.

When the server starts running as shown in Figure 11, it can now accept multiple client requests on port 1500, e.g., two clients A and B are connected to the server, as shown in Figures 12 and 13. The client A (listening port 1500, connecting port 1600) requests the server to connect with client B (on port listening port 1600, connecting port 1700) along with its certificate *ClientAPubCert.der*, after which the server requests other peer to present its certificate. The client B presents its certificate *ClientBPubCert.der* to the server. The server verifies both these certificates and notifies Auth_approved for successful authentication to both clients. The server then generates $(x.m^r, P_B)P_A$ and $(x^{-1}.m^q, P_A)P_B$ and delivers them to A and B, respectively. Client A now connects with client B to complete the process of session key establishment.

```

C:\Windows\system32\cmd.exe - java server.Server3 1500
le2\bin>java server.Server3 1500
-----
x      126382762837648723124233
x-1    35713122284198133644957795733241700133632379
n      439351292910452432574786963588089477522344331
r      603817128
q      120123346528
m      2
-----

Server Up , Listening on Port - 1500

[Client INITIATED Socket connection to the Server]
Connection Info [ Socket[addr=/127.0.0.1,port=49400,localport=1500] ]

-- INITIATE CLIENT AUTHENTICATION PROCESS BEGINS on Request FROM Client A --
1. RCD CLIENTA's PUBLIC CERT - C:/clientAPubCert.der
2. REQUESTING PEER CLIENT FOR ITS PUBLIC CERTIFICATE
3. RCD CLIENTB's PUBLIC CERT - C:/clientBPubCert.der
4. SUCCESSFULLY VALIDATED CLIENT CERT's
-- CLIENT A and CLIENT B's SERVER AUTHENTICATION COMPLETED AND SUCCESSFUL --

-- Sending Protocol Message to ClientB - <x-1.m^q, PubA>PubB --
x-1.m^q      135209001005136180916467488007952046084617757
<x-1.m^q, PubA>PubB
IDATA=15078333646922851512049129143685539769184115969897187704562506497340466117
443655968132201196001096899787034785862721872179216251208243628576;PEERKEY=19448
85247327160695029700654657129792088408325041038115564365085396868746716413167426
87794253403698997143773677671177676092183249701765016221

-- Sending Protocol Message to ClientA - <x.m^r, PubB> PubA --
x.m^r      335288649001852815734247554859602004948229889
<x.m^r, PubB> PubA
IDATA=16589140415539281998179732586612695328706297476742711401586790986662711948
982839623596656367823930353043370841240884200693212615785253202208;PEERKEY=19679
29538419931953428757203953945379373656554906918366731197599169892669419868173918
1427232383708146065036901469530990407578703154885931341

SERVER WORK COMPLETED
Closing socket - Socket[addr=/127.0.0.1,port=49400,localport=1500]

```

Figure 11: Snapshot of Server Module

The server disconnects from the clients at protocol level, after delivering the protocol messages and records the encrypted key-parts in the database as shown in Figure 10. The time taken for establishing a session key is also printed in the Figures 11 and 12, which are 5ms and 4ms for clients A and B, respectively.

```

C:\Windows\system32\cmd.exe - java client.ClientA localhost 1500 1600

ClientA Up , Listening On Port - 1600

MESSAGE TO BE SENT TO<CONNECT;<destIp>;<destPort>>?
CONNECT;localhost:1500

ENTER THE REAL MSG
CONNECT;localhost:1700

INITIATE SERVER AUTHENTICATION
Sending ClientA's Public Certificate - C:/clientAPubCert.der
AUTH_APPROVED RCD from SERVER

-- Protocol Message rcd from Server - <x.m^^r, PubB> PubA --
<x.m^^r, PubB> PubA
APPROVED[Data=-165891404155392819981797325866126953287062974767427114015867909866
62711948982839623696656367823930353043370841240884200693212615785253202288;PEERK
BI=-19679295384199319534287577039539453793736565549069183667311197599169826694198
681739181427233283708146865036901469530990407578703154885931341]
-- Decrypted Protocol message using ClientA's PrivateKey --
ProtocolMsg          335288649001852815734247554859602004948229889
ClientBPublicKey      9182765178891

MESSAGE TO BE SENT TO<CONNECT;<destIp>;<destPort>>?
CONNECT;localhost:1700

-----
SESSION KEY
17725028581524889927104561891159098711119315

TimeTakenToCreateSessionKey
[Thu Mar 04 07:10:02 IST 2010 - Thu Mar 04 07:10:02 IST 2010] - Time taken in mi
lliseconds - 5

-----
MESSAGE TO BE SENT TO<CONNECT;<destIp>;<destPort>>?

```

Figure 12: Snapshot of Client A

```

C:\Windows\system32\cmd.exe - java client.ClientB localhost 1600 1700

ClientB Up , Listening On Port - 1700

AUTHENTICATE REQUEST RCD FROM SERVER
Sending ClientB's Public Certificate - C:/clientBPubCert.der
AUTH_APPROVED RCD from SERVER

-- Protocol Message rcd from Server - (x-1.n^q.PubA)PubB --
(x-1.n^q.PubA)PubB

[DATA=15078333646922851512049129143685539769184115969897107704562506497340466117
443655968132201196001096899787034785862771872179216251208243628576;PEERKEY=19448
8574732716069502970065465712772088408325041038115564365085396868746716413167426
87794253403698997143773677671177676092183249701765016221

-- Decrypted Protocol message using ClientB's PrivateKey --

ProtocolMsg          135209001005136180916467488007952046084617757
ClientAPubKey        87637291827651

SESSION KEY
17725028581524889927104561891159098711119315

TimeTakenToCreateSessionKey
[Thu Mar 04 07:10:02 IST 2010 - Thu Mar 04 07:10:02 IST 2010] - Time taken in mi
liseconds - 4

```

Figure 13: Snapshot of Client B

The implementation of the AK-protocol was also tested for the time taken to establish a session key with that of the Secure Socket Layer (SSL). SSL establishes a secure connection in a peer-to-peer approach while the AK-protocol establishes the secure connection between two clients mediated by the trusted server. However, our clients and the server modules were running in the same network so we could evaluate the performance of the AK-protocol with that of the SSL.

Before conducting experiments we modified the standard SSL to skip the module when the clients negotiate their cipher-suite and hardcoded it to use RSA, because our implementation was coded to function with only RSA algorithm. The time taken by the key establishment process of our protocol was calculated (excluding the authentication process) with that of the time taken by *Server_Key_Exchange* and *Client_Key_Exchange* modules (see Section 3.7.3 for more information) of SSL which used Diffie-Hellman key exchange protocol.

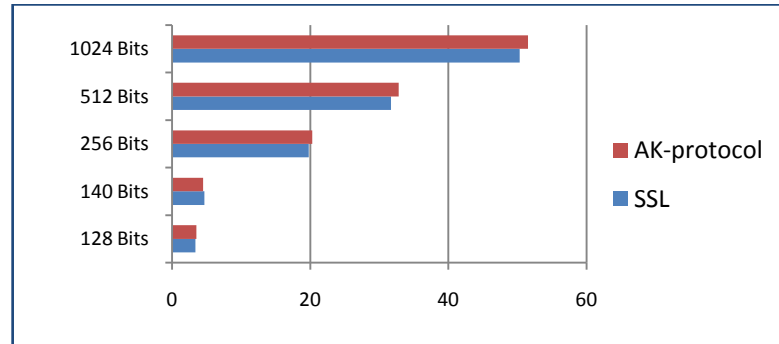


Figure 14: Protocol Analysis

We conducted experiments little less than a hundred times, while varying the encryption modulo from as small as 140 bits to 1024 bits of RSA (i.e., value of modulo η), and the parameters (exponents' r and q , multiplicative inverses (x , x^{-1}), Modulo N and its respective generator m) used in the AK-protocol. For the Diffie-Hellman key Exchange in the SSL implementation we used the same modulo and generator as in the implementation of the AK-protocol. Nevertheless, our implementation showed us comparable results with that of SSL. The time taken by two clients to establish a key between (in ms) by the Java implementations of AK-protocol and SSL is shown in Figure 14, where the average time taken is 19.411ms and 18.568ms, respectively. However, the difference between SSL and the time-taken in our implementations was more with RSA-512, RSA-1024 bits of encryption.

While our experiments it was noted that with increasing number of clients (i.e., increasing number of threads) connecting to the trusted server, the time taken by the protocols (SSL and AK-protocol) increased steeply, as shown in Figure 15. For this experiment, we kept RSA-140 bits as our encryption standard and varied the number of clients requesting trusted server to establish a session key between them.

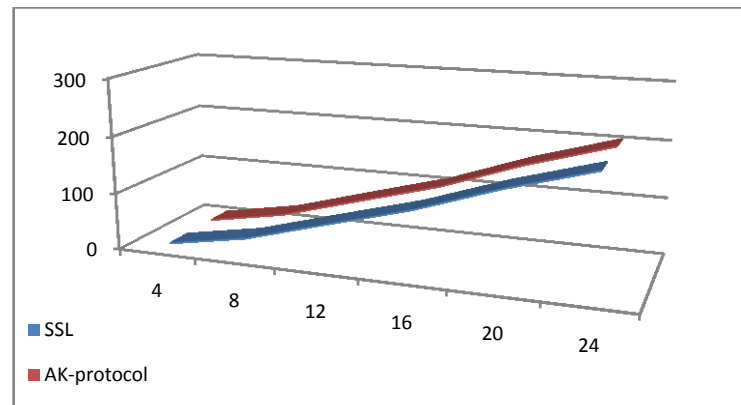


Figure 15: Protocol Analysis with increasing threads

Although, we foresee that in a practical situation of distributed environment the time taken will also depend upon the availability, promptness in responding to the incoming requests and the speed of processing computations that a server can offer.

6.3 Integrating the AK-protocol with 3Dsecure™ Model

Electronic commerce has rapidly gained momentum since early nineties, and has been equally appealing to on-line merchants, consumers and payment system providers. To reduce the fraudulent use of credit cards and increase traceability of the transactions, three-Domain Secure (3D Secure™) [3-D Secure] model, was designed which provides the issuers with the ability to authenticate cardholders during an online purchase. It is the most popular existing solution in the card payment schema developed by VISA and MasterCard, which is based on the ideas of iKP [Bellare09] and SET [Giampaolo03]. VISA and MasterCard have licensed this protocol and

many vendors have deployed it, so 3D Secure™ is considered a standard for authenticated payment.

The AK-protocol is compatible with the existing card-based business models and payment system infrastructures since the standard 3D secure protocol is a three party architecture, i.e., *a buyer, a merchant and an acquirer gateway* (an intermediary between the electronic payment world and the existing payment infrastructure, who authorizes the transactions). Hereafter, we will refer to the acquirer gateway as simply the acquirer.

6.3.1 3-D Secure Model

The 3-D secure model divides the payment system into: Issuer Domain, Acquirer Domain and Interoperability Domain. Figure 16 represents the Domain model and the principal flows in the payment protocol [Carbonell|09].

1. The **issuer domain** consists of the *Cardholder*, an *Issuer* and a component called *Access Control Server (ACS)* which authenticates cardholders during online purchases. This domain is responsible for managing the enrolment of their cardholders in the service.
2. The **Acquirer domain** consists of *Merchant*, *Acquirer* and a component *Merchant Server Plug-in (MPI)* which processes the authenticated transactions. This domain is responsible for defining the procedures to ensure that merchants participating in the Internet transactions are operating under a merchant agreement with the Acquirer.
3. The **Interoperability Domain** consists of *Directory Server (DS)* which handles all the communication between Merchant and the appropriate

ACS during request process, and *Authentication History Server (AHS)* which stores the messages from the ACS for each attempted authentication, which would assist acquirers and issuers in the event of disputes.

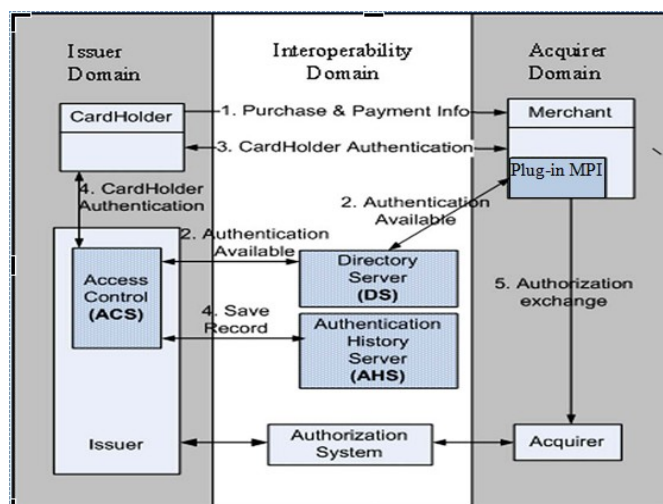


Figure 16: 3D Domain of Visa

6.3.2 3-D Secure Protocol Messages

The 3-D secure protocol messages (VEReq, VERes, PAREq and PAREs) and its purpose are mentioned in Table 15.

1. VEReq – Message from MPI to the DS or from DS to the ACS, enquiring whether authentication is available for a particular card number.
2. VERes – Message from the ACS or the DS, informing the MPI whether authentication is available for that particular card number.

3. PAREq – Message request sent from the MPI to the ACS (via the cardholder browser), to issuer to authenticate its cardholder.
4. PAREs – Message formatted, digitally signed, and sent from the ACS to the MPI (via the cardholder browser) providing the results of the issuer's 3D Secure cardholder authentication.

| | |
|--|---|
| VEReq | Acquirer BIN (Bank Identification Number), Merchant ID, Card Number, etc. |
| VERes | Cardholder enrolment status, URL of the appropriate ACS |
| PAREq | Acquirer BIN and Merchant ID From VEReq, Merchant Name Merchant name on Authentication Request Page, Merchant Country Code, Country Code of the Merchant, Merchant URL Fully qualified URL of merchant website, Purchase Amount Total amount of all purchase (items through different providers), Order Description, Brief description of items purchased. |
| PAREs | Results of cardholder authentication digitally signed by ACS. |
| Table 14: 3D Secure Protocol Messages | |

6.3.3 Process of Payment in 3-D Secure

As shown in Figure 16, the payment process in 3-D secure protocol is as follows:

1. First, the cardholder indicates the decision to buy, sending the purchases and payment information. At this moment, MPI software is activated.
2. The MPI sends a message (VEReq) to the DS to determine whether authentication services are available for the cardholder.

- a. If the cardholder is enrolled and authentication is available, the response message (VERes) instructs the MPI on how to contact the ACS (protocol continues with step 3).
 - b. If the account number of the cardholder falls outside of participating card ranges, the merchant proceeds with a standard authorization request.
3. The MPI sends an authentication request (PAREq) to the ACS. This is usually sent via the cardholder browser.
4. The ACS authenticates the cardholder by causing an authentication dialog to be displayed to the cardholder asking for a password, or by some other authentication method, such as a Visa chip card. The ACS formats and digitally signs the authentication response (PAREs), then returns it to the MPI. At this point all the details of the transaction are saved in AHS.
5. If the authentication response indicates successful authentication, the merchant forwards an authorization request with the requisite data to its acquirer for submission into an authorization system.

6.3.4 Integrated with the AK-protocol

This section illustrates the merger of the AK-protocol with the existing 3-D secure protocol. We have kept in mind that 3-D secure is already widely deployed therefore major changes would imply modifying the entire architecture. Therefore we are suggesting minor changes that would not change the architecture per se. However, we are persuaded to change the

protocol messages to benefit from our protocol, as shown in Table 16. In the rest of the section we refer to 3-D secure protocol as standard.

| | |
|--|---|
| VEReq' | Same as Standard + $S_{\text{cardholder}}\{\text{Card Number, brief purchase detail}\}$ |
| VERes | Same as Standard |
| PAReq' | Same as Standard + $S_{\text{cardholder}}\{\text{Card Number, brief purchase detail}\}$ |
| PARes' | Same as Standard + $\{\text{Key-part}\}P_{\text{Merchant}}$ |
| Table 15: Protocol Messages in our Scheme | |

The gist of the three-party authentication key establishment remains the same where ACS is acting as a trusted server from the issuer domain. As in the standard, we sought to have a component from issuer's domain, to verify the credentials that were supplied by them during system initialization. As per the standard, where two-way entity authentication is practiced, so it is in our scheme. The standard works over SSL and mutual authentication is achieved by exchanging certificates; we advise the authentication is accomplished in a similar fashion before ACS delivers the key-parts to the cardholder and the merchant. Then the cardholder and the merchant exchange their key-parts to have a shared secret. This shared secret (session key) is used to encrypt the card details from cardholder's end and the receipt from the merchant's end.

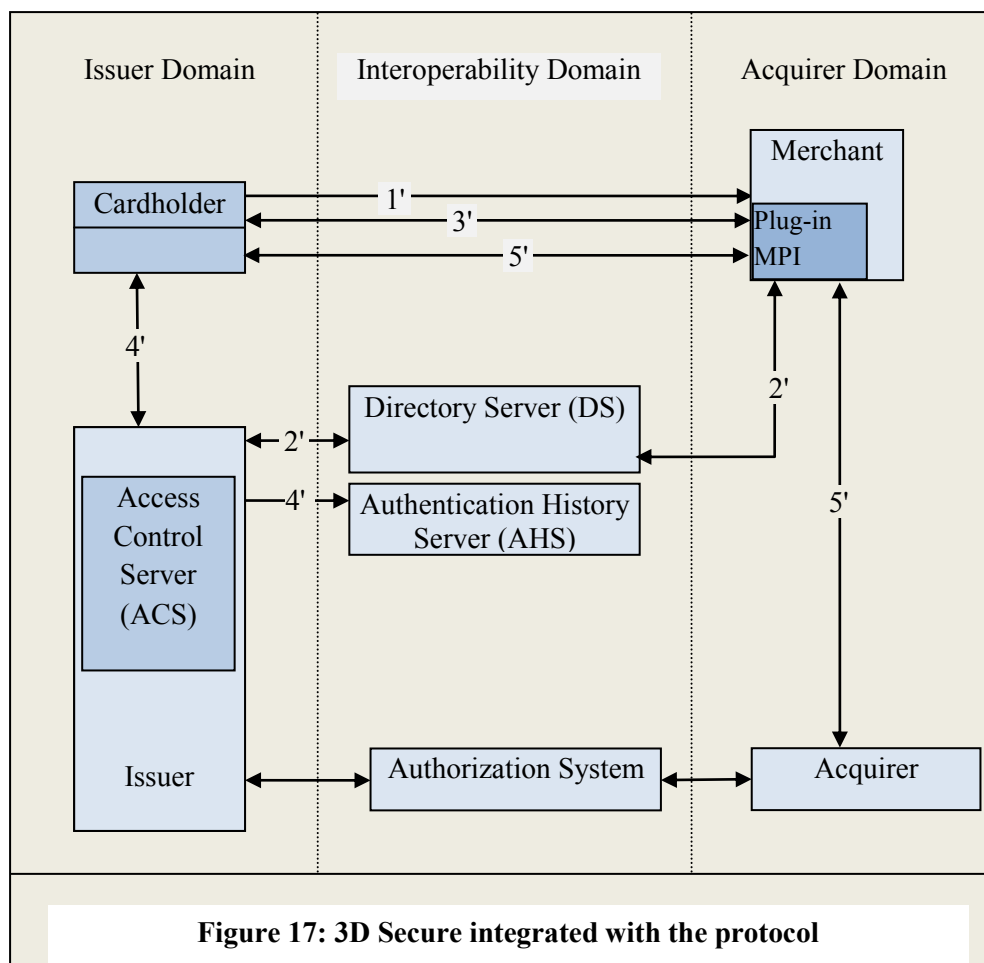
The modified steps for process of payment, as shown in Figure 17, according to the AK-protocol are as follows:

1'. When the cardholder decides to purchase, MPI is activated. The card holder now signs the brief purchase details, e.g., total amount, and his card number and sends it to MPI. Though, we restrict the cardholder to convey the card details to the merchant at this stage. Steps 2' and 3' are same as

steps 2 and 3 except that instead of protocol messages VEReq and PAReq, VEReq' and PAReq' is communicated.

4'. The ACS authenticates the cardholder as per the standard. The cardholder has its key pair (public and private keys and/or certificates) issued by its issuer during the time of initialization, which can be used to derive some new schemes of asymmetric key (like ID-based) authentication, rather than just an ATM-PIN based authentication. However, for the sake of suggesting only minor modifications we let the ACS authenticate the same way as it is in the standard. After successful authentication, ACS then delivers encrypted key-part $((x^{-1}.m^q)P_{\text{cardholder}})$ to the cardholder. The ACS formats and digitally signs the authentication response (PARes' including the key-part $((x.m^r)P_{\text{merchant}})$ for the merchant), then returns it to the MPI. ACS advises AHS to store all the relevant details at this stage.

5'. Merchant and cardholder exchange their key-parts to generate authenticated session key. Cardholder encrypts his card details using this session key and sends it to the merchant. Merchant sends an encrypted receipt of the transaction. Merchant decrypts the card details and forwards an authorization request with the requisite data to its acquirer for submission into an authorization system.



Below are the protocol message flows as per our scheme:

- 1'. Cardholder \rightarrow Merchant: $\{\text{Cardnumber}, \text{BriefPurchaseInfo}\}_{S_{\text{Cardholder}}}$
- 2'. MPI \rightarrow DS: VEReq'
 DS \rightarrow MPI: VERes
- 3'. MPI \rightarrow ACS: PAReq' (ACS authenticates credentials of the Merchant)

4'. ACS \rightarrow Cardholder:

- a. Authentication Challenge
- b. Delivers $((x^{-1}.m^q)P_{\text{cardholder}})$ after successful authentication.

ACS \rightarrow Merchant: PAREs'

ACS \rightarrow AHS: Store all the relevant information into the database.

5'. Cardholder \leftrightarrow Merchant: (a) Compute session key m^{r+q} .

Cardholder \rightarrow Merchant: $(\text{CardDetails})m^{r+q}$

Merchant \rightarrow Cardholder: $(\text{Receipt})m^{r+q}$

Merchant \gg Acquirer: CardDetails, and

$\{\text{Cardnumber}, \text{BriefPurchaseInfo}\}S_{\text{Cardholder}}$

6.3.5 Analysis of the Proposal

In this section, we analyse the outcome of the integration of 3Dsecure and the AK-protocol. We have suggested that cardholders are provided with the key pairs, and/or certificates by their issuer, which can be used to derive new scheme for entity authentication that will serve something more than just an ATM PIN for identifying a cardholder. However, we are still considering the analysis of the existing system where a cardholder is authenticated by an ATM PIN number.

After authenticating cardholder and merchant, ACS who acts as a trusted server in our case, distributes key-parts for them to generate a session key.

This provides a secured service after both cardholder and merchant are authenticated. This extra layer of security helps in the following points as pointed out by Murdoch and Anderson [Murdoch-Anderson|10]:

- As per our scheme, Cardholders use their private key for signing their card number and brief purchase details which prevents fraudulent cardholders from creating fake request and also ensures that the cardholder actually possesses a valid certificate and the correct private key. This added mechanism also prevents *replay attacks* since the adversary cannot replay the old signed card numbers (or just the plain card numbers as per the standard) to a merchant, claiming to be a benign card holder.
- It is hard for users to differentiate between the legitimate Verified by Visa pop-up window or inline frame, or fraudulent phishing site. The added mechanism in our scheme gives opportunity to both cardholder and merchant to share a secret that can be used to secure vital credentials or information after they are identified by the ACS.
- As per the standard, in cases when a client is not able to validate the public key certificate of the merchant the process either aborts or is carried out without certificate verification. Our scheme assists cardholder and merchant to build up level of trust as ACS (who is from the issuer domain) takes charge of authenticating the merchant and the cardholder. They can be sure that the authentic session key will be generated by only authentic merchants and cardholders, since ACS generates and distributes the ephemeral keys only to identified merchant and cardholder.

- In our scheme, the cardholder is asked to sign the purchase detail and card number in the first step which provides two benefits: (a) it prevents the merchant from changing the purchase details later, since merchant sends the same signed value in the PAREq' and later to the authorization system; and (b) it engages the client to some computations which helps in mitigating *Distributed Denial of Service (DDoS) attacks* where an adversary can otherwise flood the server with a number of fake client requests.
- As per our scheme, the security of entire financial transaction does not depend only on the password that was created with the issuer at the time of initialization, which is prone to online and offline dictionary attacks. It also depends upon the dynamically generated session key coordinated by the ACS, cardholder and merchant.
- Usually, an adversary takes control of the system or hijacks a session after the clients have achieved authentication, but our scheme mitigates such attempts since the vital credentials are encrypted by the dynamically generated session key.
- The cardholder gets an encrypted receipt; similarly the merchant gets encrypted card details, which can be used as a reference for them to resolve disputes. AHS also stores the encrypted key parts, a signed copy of purchase details and the card number, and other relevant details all indexed with a unique transaction ID for later reference especially for the cases of non-repudiation as discussed in Section 5.8.11. The AK-protocol also offers the key escrow provisions as discussed in Section 5.8.13.

6.4 AK-protocol in Wireless Environment

In this section we will discuss the deployment of the AK-protocol for wireless environment (a) for mobile payment system, and (b) securing real-time communication using stream cipher algorithm RC4.

6.4.1 Standard Mobile Payment System

The Trusted Computing Group (TCG) [TCG] has defined a set of hardware and software specifications for Trusted Computing Technologies. In a standard mobile payment scheme three parties participate: a *mobile device* which represents a user, a *merchant*, and a *financial service provider*, e.g., a bank or credit card service provider. There is another element, Trusted Third Party (TTP) who authenticates the authorized users, however in practice TTP is part of the financial service provider.

There are two types of e-payment applications: *Check-like* and *Cash-like* payments. *Check-like payments* require a certain amount of virtual money which is debited from the customer before the actual payment whereas *Cash-like payments* require the involvement of customer's account in each transaction. Since the latter is much more popular nowadays, we focus on this type of payment [Anderson|95].

A customer needs to register with the TTP (or financial institution) and share his account information before being able to get the payment service. The client¹¹ is also required to store payment information on their device, e.g., their credentials (private key and the client account information), corresponding bank account and other billing information, in a secured

¹¹ A client is a registered user possessing a pair of keys and certificate if required.

component called *e-wallet*. The e-wallet initialization aims to generate a public/private key pair and securely stores the private part into the mobile device which is later used for authentication in payment transactions.

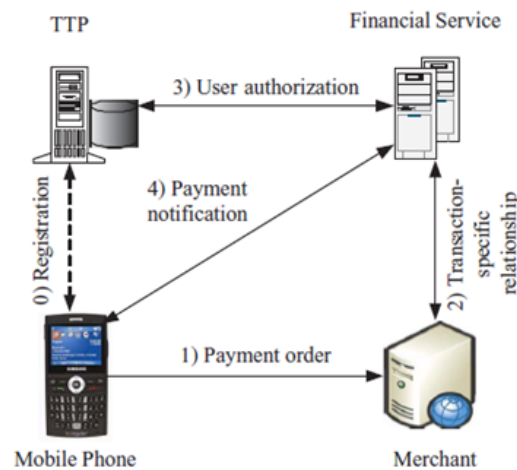


Figure 18: Standard Mobile Payment System

As shown in Figure 18, when a client initiates a payment order to the merchant (1), it invokes payment service which forwards the request via payment gateway to the bank (2). The bank then forwards received request to TTP who eventually authenticates and authorizes the customer (3). If the customer is successfully authenticated and authorized, his account is debited. The payment transaction is complete when a notification of payment (e.g., a printed receipt or e-receipt) is presented to the customer and the merchant (4).

In the standard architecture we do not authenticate the merchant before he is authorized to receive the card details. However, considering the advent of mobile fraud we think that it is as important to authenticate the entity that is receiving the payment as it is to authenticate the entity that is paying.

6.4.2 Integration of Mobile Payment System and the AK-protocol

At the time of system initialization, TTP or issuer generates key pairs for the individual clients and respective public key certificate which is securely stored into an e-wallet, and its access would require the client to enter his password. A merchant is a special client registered under the system and possesses key pairs and public key certificates.

1. $TTP \rightarrow \text{Client A: } \{e\text{-wallet}_A\}$
2. $TTP \rightarrow \text{Merchant M: } \{e\text{-wallet}_M\}$

Below is the process of authentication when client A sends a request to TTP to authenticate the merchant, along with its public key certificate P_A . Steps 2 and 3 can be optional for the typical systems that do not need to authenticate a merchant, but we advise to include them to increase the client's level of trust.

1. $A \rightarrow TTP: P_A, \text{Req Merchant}$
2. $TTP \rightarrow M: \text{Req Cert Merchant}$
3. $M \rightarrow TTP: P_M$

The process of authentication of the participating clients is accomplished only once when they enable the feature of “secured services” in their mobile devices (for more details see Section 5.7). After successful authentication of the merchant and the client they are eligible to communicate with each other.

The TTP delivers key-parts to the clients (steps 1 and 2). After the exchange of protocol messages a session key is established at both ends (steps 3 and

4). The client sends encrypted account details to the merchant and merchant sends encrypted receipt to the client (steps 5 and 6). These added steps would ensure greater level of trust between the client and the merchant where the system has the capability for non-repudiation.

1. $TTP \rightarrow A: (x.m^r)P_A, P_M$
2. $TTP \rightarrow M: (x^{-1}m^q)P_M, P_A$
3. $A \rightarrow M: (x.m^r)P_M$
4. $M \rightarrow A: (x^{-1}m^q)P_A$
5. $A \rightarrow M: \{\text{Account Details}\}m^{r+q}$
6. $M \rightarrow A: \{\text{Receipt}\}m^{r+q}$

This means are three rounds of protocol messages that are exchanged between the TTP and the clients. Keeping in mind that specifically in a mobile environment (closed environment) to-and-fro communications are more time and power consuming, we think that these three rounds of protocol messages can be further reduced to two rounds. The TTP (or the base station) delivers both the key-parts to each one of them (see the steps 1 and 2 below) to cut down the round when the merchant and the clients exchange their key-parts to establish a session key. In this case, it is not required to communicate the public key of the other participant. Although, the protocol messages (Steps 1 and 2 below) contains both the key-parts but they are encrypted with the public key of client A and merchant, respectively. An adversary who has captured the protocol messages would require efforts as stated in Equation b.2 to recover the session key.

1. $TTP \rightarrow A: (x.m^r)P_A, (x^{-1}m^q)P_A$
2. $TTP \rightarrow M: (x.m^r)P_M, (x^{-1}m^q)P_M$
3. $A \rightarrow M: \{\text{Account Details}\}m^{r+q}$
4. $M \rightarrow A: \{\text{Receipt}\}m^{r+q}$

However, if TTP provides the both key-parts to the client and merchant then it effectively eliminates one of the key benefits of the protocol where trusted server provides different key-parts to the participating parties which they exchange to establish a session key. This could be a conscious trade-off to be made while initializing a system.

The benefits of integrating the AK-protocol with the standard mobile payment systems will be the same as those discussed in the Section 6.3.5.

6.4.3 Real-Time secured communication with AK-protocol

When two mobile clients or any two devices in a wireless network, want to have a real-time secured communication, e.g., secured voice communication, we follow the same process as shown in the previous section to establish a shared secret between the two entities. Once the participants have established the session key, a fast encryption/decryption algorithm¹² is deployed that can deal with the real-time data.

Implementations of algorithms, AES, DES, and 3DES on GSM (Global System for Mobile communications) were studied by Sachin and Kumar [Sachin10]. However, it is still an open problem to find ciphers that are both secure and have a high encryption speed. Since, the block ciphers are known to have a high level of security but known to have a relatively slow E/D [Encryption/Decryption] rates. Stream ciphers, on the other hand, are known for their simple structure and hence their very high E/D

¹² Stream ciphers are an important class of encryption algorithms which encrypts individual bits or bytes of the plaintext message one at a time. Block ciphers encrypt a fixed length of plaintext bits (or bytes), at any instance. A stream cipher attempts to capture the spirit of one time pad by using a short key to generate the key stream which appears to be random.

[Encryption/Decryption] rates. But, this simple structure is usually a source of security weaknesses [Shehri|10].

RC4 Algorithm

RC4 is a stream cipher designed by Rivest for RSA Data Security which is a variable key-size stream cipher with byte-oriented operations. RC4 is used for file encryption in products such as RSA SecurPC and SSL. Riad et al. [Riad|09] have analyzed RC4, by applying the NIST suite of statistical tests and concluded that using RC4 is secure and random enough to be used within the converged network, especially for real-time applications. Therefore, we suggest the use of RC4 algorithm to secure the real-time wireless communications after having a session key established by the AK-protocol.

The RC4 algorithm is based on the use of a random permutation. A variable length key k that ranges from 1 to 256 bytes is used to populate a 256-byte state vector S , with elements $S[0] - S[255]$. The output of the algorithm is a byte k_i generated from the state vector by selecting one of the 255 entries, thereafter the state vector is permuted again. An XOR operation (\oplus) of one byte of plaintext and k_i is computed for encryption. Similarly, the cipher text is XORed with the same k_i generated at the other end for decryption.

Let's say two wireless clients M and M' want to communicate securely. The trusted server (or base station) then mediates to establish a session key between them. The wireless system uses RC4 for encryption and decryption of the real time data.

The process of encryption and decryption using RC4 is shown in Figure 19. The RC4 algorithm takes the generated session key as input to initialize the

state vector. Here K_i is the 8 bits key generated by RC4, P_i is 8 bits of plain text, $P_i \oplus K_i = C_i$ is the 8 bits of cipher text. At the other end client M' who has the same session key upon reception of C_i computes $C_i \oplus K_i = P_i$ to get the plaintext. The session key is generated by the AK-protocol, which will add to the security (which is discussed in Section 5.9.6) of RC4 encryption.

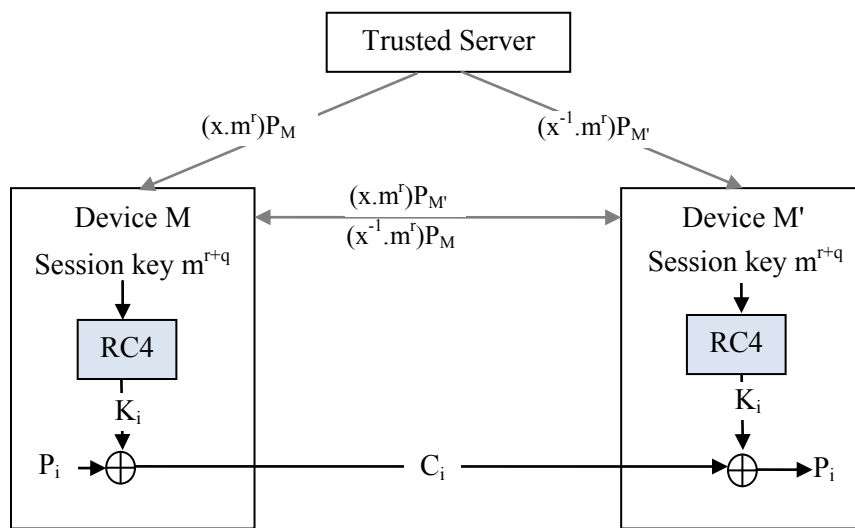


Figure 19: Encrypting Real-time Communication with RC4

6.5 Examples of the AK-protocol

This section presents three examples of the AK-protocol that can be used to understand the data flow and the mathematical computations required to establish a session key at both ends.

6.5.1 A Simple Example

A. System Initialization

1. Choose two prime numbers $P = 65521$, $Q = 32749$
2. Compute $\eta = P \cdot Q = 2145747229$
3. Compute $\Phi(\phi) = 65520 \cdot 32748 = 2145648960$
4. Choose N for modular operations, $N = 65537$, Generator $m = 3$.
5. Compute set of pairs of multiplicative inverses: $(23, 45591)$, $(27, 24273)$, $(61, 56942)$, $(89, 45655)$, $(127, 39219)$, etc.

B. Key generation

6. Generate key pairs for the trusted server and clients say A and B.

| | Server | Client A | Client B |
|-------------|------------|-----------|------------|
| Public Key | 11 | 17 | 19 |
| Private Key | 1560471971 | 883502513 | 1242217819 |

C. Real time Computations

C.1 Server end

1. Choose $(x, x^{-1}) = (23, 45591)$, $r = 6$, $q = 9$
2. Compute $x.m^r \bmod N = 23 \cdot (3^6) = 23 \cdot 729 = 16767 \pmod{65537}$
3. Compute $x^{-1}.m^q \bmod N = 45591 \cdot (3^9) = 45591 \cdot 19683 \pmod{65537} = 897367653 \pmod{65537} = 35049 \pmod{65537}$
4. Encrypt key-part for client A $\{x.m^r\}P_A \bmod \eta = (16767^{17}) \pmod{2145747229} = 2016996287$
5. Encrypt another key-part for client B $\{x^{-1}.m^q\}P_B \bmod \eta = (35049^{19}) \pmod{2145747229} = 1001665371$

C.2 Client end

Decrypt the message received to get the key-part and then encrypt it with the public key of the other client and exchange them.

1. $\{x.m^r\}P_B \bmod \eta = (16767^{19}) \pmod{2145747229} = 784860764$
2. $\{x^{-1}.m^q\}P_A \bmod \eta = (35049^{17}) \pmod{2145747229} = 929278930$

After the exchange both clients have both key-parts,

3. Compute $\{(x.m^r).(x^{-1}.m^q)\} \bmod N = 16767 \cdot 35049 \pmod{65537} = 587666583 \pmod{65537} = 61841$ is the session key established at both ends.

6.5.2 Example RSA-123 Bits

A. System Initialization

1. Choose two prime numbers $P = 108086391056891903$, $Q = 55340232221128654847$
2. Compute $\eta = P \cdot Q = 5981525981032121372618813159076003841$
3. Compute $\Phi(\phi) = 5981525981032121317170494546890457092$
4. Choose N for modular operations, $N = 864608136454559457049$,
Generator $m = 2$.
5. Compute pair of multiplicative inverse $(61938743289, 78818796327270573902)$

B. Key generation

6. Generate key pairs for the clients A and B.

Client A

Public key = 94729836477

Private Key = 2420924935109769594914530477100477645

Client B

Public key = 29137283681

Private Key = 4153442163589190600456383648658904621

C. Real time Computations

C.1 Server end

1. Choose(x, x^{-1}) = (61938743289, 78818796327270573902), $r = 389393$, $q = 262594058$.
2. Compute $x.m^r \bmod N = 326753863534053862509$
3. Compute $x^{-1}.m^q \bmod N = 642251862079891166423$
4. Encrypt key-part for client A $\{x.m^r\}P_A \bmod \eta = 4295081288836038820259622745838051087$
5. Encrypt another key-part for client B $\{x^{-1}.m^q\}P_B \bmod \eta = 3827657607419813442816253956438349973$

C.2 Client end

Decrypt the message received to get the key-part and then encrypt it with the public key of the other client and exchange them.

1. $\{x.m^r\}P_B \bmod \eta = 3624236233461722317643658250803545936$
2. $\{x^{-1}.m^q\}P_A \bmod \eta = 2448537185616354252124524736201221866$

After the exchange both clients have both key-parts,

3. Compute $(x.m^r).(x^{-1}.m^q) \bmod N = 437872649104419273489$ is the session key established at both ends.

6.5.3 Example with RSA-140 Bits

A. System Initialization

1. Choose two prime numbers

$P = 3398717423028438554530123627613875835633986495969597423$
 $490929302771479,$

$Q = 62642001874012850961516549482644422193020371786235090191$
 11660653946049

2. Compute $\eta = P.Q = 21290246318258757547497882016271517497806$
 $703963277216278233383215381949984056495911366573853021918316$
 $783107387995317230889569230873441936471$

3. Compute $\Phi(\phi) = 2129024631825875754749788201627151749780670$
 $396327721627823338321538194032113888548164292317124334243846$
 $5052451971642637783126628283485218944$

4. Select N for modular operations,
 $N = 439351292910452432574786963588089477522344331$ (149 bits
prime number), Generator $m = 2$.

5. Compute pair of multiplicative inverse
 $(126382762837648723121233,$
 $357131922284190133644957795733241700133632379).$

B. Key generation

6. Generate key pairs for the clients A and B.

Client A

Public key A = 87637291827651

PrivateKeyA = 1251184752043678012028463090373152812908238

812245877317213470713707291834164096168783493507528991733479760
2927506535441842247514066403828587

Client B

Public key B = 9182765178891

PrivatekeyB = 16863541073854604104224190748904315131079
101452569987390188998913568600249655987410876231036397041128111
884035689898898244741646177597746339

C. Real time Computations

C.1 Server end

1. Choose $(x, x^{-1}) = (126382762837648723121233,$
357131922284190133644957795733241700133632379), $r = 603817128,$
 $q = 120123346528.$
2. Compute $(x.m^r) \bmod N$

= 135209001005136180916467488007952046084617757
3. Compute $x^{-1}.m^q \bmod N$

= 335288649001852815734247554859602004948229889
4. Encrypt key-part for client A $\{x.m^r\} P_A \bmod \eta$

= 97661883910802800553178434941859206773874268541723758
09052704380183341146234174521953626091225617479492166890
021432473553755308717725202859

5. Encrypt another key-part for client B $\{x^{-1}.m^q\}P_B \bmod \eta$

$$= 19401483998169356836829851292499558296979148348519280 \\ 022607267202433943439070051329346478339568715493231719470759 \\ 708741332837316989947334507$$

C.2 Client end

Decrypt the message received to get the key-part and then encrypt it with the public key of the other client and exchange them.

a. $\{x.m^r\}P_B \bmod \eta$

$$= 15078333646922851512049129143685539769184115969897107704562 \\ 506497340466117443655968132201196001096899787034785862771872 \\ 179216251208243628576$$

b. $\{x^{-1}.m^q\}P_A \bmod \eta$

$$= 16589140415539281998179732586612695328706297476742711401586 \\ 790986662711948982839623696656367823930353043370841240884200 \\ 693212615785253202288$$

After the exchange both clients have both key-parts,

c. Compute $\{(x.m^r).(x^{-1}.m^q)\} \bmod N$

$$= 17725028581524889927104561891159098711119315 \text{ is the session key established at both ends.}$$

Chapter 7 Conclusions and Further-Work

7.1 Summary of the Thesis

In this thesis we have seen some, out of an assortment of key establishment protocols that exist in the literature. We have also seen the benefits of having a trusted server to mediate the key establishment between two entities. Practically as agreed by the larger audience, in a distributed network asymmetric-key based authentication schemes offer better solutions to scalability, mobility, reusability and ease in key distribution. They also provide credentials for authentication where high valued critical data are communicated. For this reason, we have suggested use of certificates for authentication in our protocol. However, the AK-protocol is open to trade certificate-based authentication schemes for any identity-based scheme, when medium level of security is required.

Having a trusted server to authenticate the participating clients offers a more reliable medium for certificate verification of the certificates generated by a Certification Authority. Furthermore, the trusted server also generates the

ephemeral keys during the process of key establishment which brings a higher level of trust to all the entities subscribed to the system.

The architecture of the AK-protocol is described in Section 5.5, where the participating clients present their certificates to authenticate their identity before they can communicate with other clients. The trusted server not only mediates the process of authentication of the participating clients, but also generates a pair of distinct but related ephemeral keys for the process of key establishment. The clients have to exchange the received ephemeral keys with the other participating client to establish a common session key which is used to secure further communications. All the protocol messages between the server-to-clients and client-to-client are encrypted using RSA algorithm.

We examined the AK-protocol in Section 5.8 looking for the general security properties as expected by any key establishment protocol and found that the AK-protocol satisfies the following properties:

- a. *Perfect Forward Secrecy*, as per our protocol the session keys are not saved by the clients and are deleted from their system buffers after use, while they save the transaction ID for future reference. For the system to offer non-repudiation, the trusted server stores the protocol messages $(x.m^r)P_A$ and $(x^{-1}.m^q) P_B$ encrypted with two different public keys in a secured database. As discussed in Section 5.8.1 an adversary needs to accomplish two tasks to get the old session keys: (a) hack into the secure database of trusted server and retrieve two encrypted key-parts, (b) acquire private keys of both participating clients.

- b. *Known Session-Key Resilience*, the trusted server makes use of three random parameters r , q and (x, x^{-1}) each time it generates the ephemeral keys, which makes it unpredictable. As discussed in Section 5.9.3, Case (e), an adversary cannot impersonate as a benign client even after acquiring his certificate.
- c. *Unknown Key-Share Resilience*, even if an adversary tricks both clients to share their protocol messages (which are encrypted ephemeral keys), the bit complexity to recover the session key would be $O(\eta \cdot (2 \cdot (\log_m N)^3))$, for more details see Section 5.9.2.
- d. *Key-Control*, the trusted server generates two *different but related* ephemeral keys for the participating clients after which the clients have to exchange the ephemeral keys they possess to complete the key establishment process. The ephemeral key generation is entirely under the control of the trusted server while the key establishment process requires all three parties, as discussed in Section 5.8.4.
- e. *Key Freshness*, the element of freshness is added by three randomly chosen parameters r , q and (x, x^{-1}) to compute the protocol messages, see Section 5.8.5.6 for more details.
- f. *Bandwidth Required*, the number of bits in the protocol messages from server-to-clients and client-to-client is 2η and η , respectively. However, as discussed in the Section 5.9.4, when a signed hash of each protocol message is sent along in the server-to-clients protocol messages then the bandwidth increases to 3η while it remains same for client-to-client messages.
- g. *Scalability*, as per our scheme either a CA or a client takes charge of key generation, later the trusted server is updated with the database

of client's credentials to authenticate communicating clients, this makes the system highly scalable, see Section 5.8.8 for more details.

- h. *Ease in Key Distribution and Directory Service*, having a centralized server that stores the public key certificates of all the subscribed clients in a secure database, provides promising capabilities for key distribution and directory services see sections 5.5.9 and 5.5.10 for more details.
- i. *Non-Repudiation*, we suggested in Section 5.8.11 that trusted server may save the encrypted key-parts and other relevant transaction related information in a database, indexed with a unique transaction ID (clients save the transaction ID for reference), that would assist in resolving disputes.
- j. Additionally, we demonstrated in the Section 5.8.5, that the AK-protocol satisfies all the desired properties in a 3PAKE as suggested by Chen et al. [Chen-Lee08].
- k. Key Escrow, in the Section 5.8.13, we have suggested that the CA possesses the private keys of the clients and could request the trusted server to retrieve the information associated with a particular transaction ID, to allow law enforcement agencies to decrypt the cipher text when warranted.

However, we have also found that our protocol relies on keeping the secrecy of the long term private key of individual entities. When an entity's long term private key is compromised and the adversary who now possesses that particular private key also captures all the protocol messages communicated between server-to-clients and client-to-client, he could eventually recover the session key. We advise that the liability of the private key remains with

the entity or the issuer and in the event of key compromise they should exercise generation of new set of keys and certificates.

The AK-protocol was also analyzed for different applicable attack scenarios in a distributed environment which are summarized below:

- a. *Replay attack*, in Section 5.9.1 we discussed three scenarios when an adversary records the protocol messages to replay them later, and found that the system can cope in those situations.
- b. As discussed in Section 5.9.3, the AK-protocol is resilient to *impersonation attacks*.
- c. *Distributed Denial of Service attacks*, we have examined the AK-protocol DDoS resilient properties as suggested by Stebila and Ustaoglu [Stebila-Ustaoglu|09] and J. Smith and et al. [Smith-Tritilanunt|07]. We have proved in Section 5.9.5 that the protocol is resilient to memory, computational, transmission and flooding DDoS attacks.
- d. Furthermore, as discussed in Section 5.9.4 we also examined a situation where an attacker can craft protocol messages to mislead the clients and proposed two mechanisms to prevent messages from an adversary pretending to be the trusted server (a) by exercising the two-way authentication, where not only clients but the server also presents its certificate for verification; or (b) the trusted server creates the hash of the message and signs it for the clients to know that messages are actually coming from an authentic server.

We have computed the bit complexity in Section 5.9.2 and efforts required to perform the cryptanalysis in Section 5.9.6 which demonstrates that mathematically AK-protocol is robust.

With regards to the practical applicability of the AK-protocol, we have dedicated *Chapter 6* to different aspects of its possible executions. Following is a brief summary of the demonstrated deployments:

- (a) In Section 6.2, the multithreaded Java implementation of the AK-protocol on TCP layer based on multiple-clients and single-server architecture is explained. We also conducted some experiments to show that the time taken by the AK-protocol is comparable with that of SSL when the trusted server and the participating clients were in the same network.
- (b) We discussed the possibility of integrating the AK-protocol with the existing 3-D secure protocol of MasterCard and Visa for online payments systems. Since 3-D secure is widely deployed we have suggested minor modifications (for more details see Section 6.3.4) to the messages VEReq, PAREq and PAREs, of the standard protocol. We have studied the flaws in the standard protocol as pointed out by Murdoch and Anderson [Murdoch-Anderson|10] and have modified few steps in the standard to benefit from the protocol: (i) Cardholders sign the brief purchase details and their card number with their private keys; (ii) after ACS authenticates both merchant and cardholder it delivers the encrypted key-parts for them to establish a session key; and (iii) the merchant and cardholder exchanges the card details and receipt encrypted with this dynamically generated session key. We have also evaluated the

outcome of this modified version which offers promising results over the standard protocol as discussed in Section 6.3.5.

- (c) Two aspects of implementing the AK-protocol in the wireless environment were suggested: First, integration with the standard mobile payment systems. As explained in the Section 6.4.2, each mobile user secures the vital credentials in an e-wallet which is used during authentication. After having a session key established between the merchant and cardholder they exchange the account details and receipt, encrypted with the session key. Second, for securing real-time communication between two users, where generation of session key is accomplished by the AK-protocol and then a fast stream cipher algorithm like RC4 is used for encryption/decryption of the real-time data as illustrated in the Section 6.4.3.
- (d) For the sake of understanding the data flow and the cryptographic mathematics involved in the AK-protocol we have presented three examples in the Section 6.5.

Thus, we see that the deployment of the AK-protocol is practical in a wide arena.

7.2 Further Work

As discussed in the Section 6.3.4 we have suggested an integration of the AK-protocol with 3-D Secure, it could be actually deployed in a prototype

system and various experiments could be performed to analyse the new system against the standard system.

In Section 6.4.3, we have suggested an implementation for wireless environment when the AK-protocol is used for key establishment and RC4 as encryption/decryption algorithm. A simulation of this scheme could be designed and experiments could be conducted to analyze it, specifically for its speed. The simulation could have some mobile devices with embedded applications that generates session key using the AK-protocol. The simulated base station could act as trusted server which accomplishes the process of key establishment between two mobile devices in a test network. The examples presented in Section 6.5 could assist in computing the mathematical values and understanding the data flow of the AK-protocol.

References

- [3-D Secure] 3-D Secure™. “System Overview”. 70015-01 External version 1.0.2 May 01, 2003. Copyright 2002-2003 Visa International.
- [Abdalla|05] M. Abdalla, P. A. Fouque and D. Pointcheval. “Password-based authenticated key exchange in the three-party setting”. In the proceedings of PKC’05, LNCS, vol. 3386, 2005, pp. 65-84.
- [Aggarwal|09] D. Aggarwal and U. Maurer. “Breaking RSA Generically is Equivalent to Factoring”. In the proceedings of EUROCRYPT 2009, pages 36-53.
- [Aivaloglou|08] Efthimia Aivaloglou, Stefanos Gritzalis and Charalabos Skianis. “Trust establishment in sensor networks: behaviour-based, certificate-based and a combinational approach”. International Journal of System of Systems Engineering, Volume 1, Number 1-2/2008, pages 128 – 148.
- [Anderson|95] R.J. Anderson, S.J. Bezuidenhout. "Cryptographic Credit Control in Pre-payment Metering Systems". In the proceedings of IEEE Symposium on Security and Privacy 1995, pages 15-23.

- [Bellare|00] M. Bellare, D. Pointcheval and P. Rogaway. "Authenticated key exchange secure against dictionary attacks". In the proceedings of Eurocrypt'00, LNCS, vol. 1807, 2000, pages 139-155.
- [Bellare|09] Mihir Bellare et al. "Design, Implementation and Deployment of the iKP Secure Electronic Payment System". IEEE Journal of Selected Areas in Communications, VOL 18, NO. 4, April 2000.
- [Bellovin|92] S. M. Bellovin and M. Merritt. "Encrypted key exchange: password-based protocols secure against dictionary attacks". In the proceedings of IEEE Symposium on Research in Security and Privacy, 1992, pages 72-84.
- [Blakey|09] E. Blakey. "Factorizing RSA Keys, An Improved Analogue Solution". Volume 27, Number 2, pages 159-176, Feb 2009.
- [Boneh|01] D. Boneh and M. Franklin. "Identity-based Encryption from Weil pairing". In Advances in Cryptology, Crypto 2001. Lecture Notes in Computer Science, Vol. 2139. Springer-Verlag 2001 pages 213-229.
- [Boyd|00] C. Boyd and A. Mathuria. "Key establishment protocols for secure mobile communications: a critical survey". Computer Communications 23 (2000) 575-587.
- [Boyd|03] Colin Boyd and Anish Mathuria. "Protocols for Authentication and Key Establishment". Springer, 2003.

- [Boyd|96] Colin Boyd. "A class of flexible and efficient key management protocols". In the proceedings of 9th IEEE Computer Security Foundations Workshop, pages 2-8. IEEE Computer Society Press, 1996.
- [Boyko|00] V. Boyko, P. MacKenzie and S. Patel. "Provably secure password-authenticated key exchange using Diffie-Hellman". In the proceedings of Eurocrypt'00, LNCS, vol. 1807, 2000, pages 156-171.
- [Brent|00] Richard P. Brent. "Recent Progress and Prospects for Integer Factorisation Algorithms". In the proceedings of COCOON 2000.
- [Brown|06] R. L. Brown. "Breaking RSA may be as difficult as factoring". In Cryptology ePrint Archive, Report 205/380, 2006.
- [Burrows|96] Michael Burrows, Martin Abadi and Roger Needham. "A Logic of Authentication". In William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.
- [Buttyán|10] Levente Buttyán, László Dóra, Fabio Martinelli and Marinella Petrocchi. "Fast certificate-based authentication scheme in multi-operator maintained wireless mesh networks". Computer Communications, COMCOM 4144, 1 February 2010.
<http://www.ScienceDirect.com/science/journal/01403664>

- [Carbonell|09] Mildrey Carbonell, José María Sierra and Javier Lopez. “Secure multiparty payment with an intermediary entity”. *Computers & Security*, Volume 28, Issue 5, July 2009, pages 289-300.
- [Chang|04] Chin-Chen Chang and Ya-Fen Chang. “A novel three-party encrypted key exchange protocol”. *Computer Standards & Interfaces*, Volume 26, Issue 5, September 2004, pages 471-476.
- [Chen|02] L. Chen and C. Kudla. “Identity based key agreement protocols from pairings”. In the proceedings of, The 16th IEEE Computer Security Foundations Workshop, 2002, pages 219-213.
- [Chen|06] L. Chen, Z. Cheng and N.P. Smart. “Identity-based key agreement protocols from pairings”. *Cryptology ePrint Archive*, Report 2006/199, 2006. Available at <http://eprint.iacr.org/2006/199>.
- [Chen|08] H.-B. Chen, T.-H. Chen, W.-B. Lee and C.-C. Chang, “Security enhancement for a three party encrypted key exchange protocol against undetectable online password guessing attacks”. *Computer Standards & Interfaces* 30 (2008) 95-99.
- [Cheng|01] P.C. Cheng. “An Architecture for the Internet Key Exchange”. *IBM Systems Journal*, VOL 40, PART 3, pages 721-746, 2001.

- [Cheng|04] Z. Cheng, L. Vasiu and R. Comley. "Pairing-based one-round tripartite key agreement protocols". Cryptology ePrint Archive, Report 2004/079, available at <http://eprint.iacr.org/2004/079/>.
- [Cheng|05] Z. Cheng, L. Chen, R. Comley and Q. Tang. "Identity-based key agreement with unilateral identity privacy using pairings". Cryptology ePrint Archive, Report 2005/339.
- [Cheng|06] Z. Cheng and R. Comley. "Attacks on an ISO/IEC 11770-2 key establishment protocol". International Journal of Network Security, 3, pages 290-295, 2006.
- [Chen-Lee|08] T.-H. Chen, W.-B. Lee and H.-B. Chen. "A round and computation efficient three-party authenticated key exchange protocol". The Journal of Systems and Software 81, pages 1581-1590, 2008.
- [Chien|04] H. Y. Chien. "Improved ID-based Tripartite Multiple Key Agreement Protocol from Pairings". Proceedings of the 14th Information Security Conference, (2004), Taiwan.
- [Chien|04] H.-Y. Chien and T.-C. Wu. "Highly Efficient Password Based Three-Party Key Exchange in Random Oracle Model". Proceedings of IEEE ISI 2008, Taiwan, June 17-20, pages 69-76, Lecture Notes in Computer Science, Vol. 5075. Springer, Berlin.

- [Chien|09] Hung-Yu Chien and Tzong-Chen Wu. “Provably Secure Password Based Three-Party Key Exchange With Optimal Message Steps”. The Computer Journal, Vol. 52 No. 6, 2009
- [Chow|05] S.S.-M. Chow and K.-K.R. Choo. “Strongly-secure identity-based key agreement and anonymous extension”. In the proceedings of ISC’07, LNCS, vol. 4779, 2005, pages 203-220.
- [Chung|08] H.R. Chung and W.C. Ku. “Three weaknesses in a simple three party key exchange protocol”. Information Science, 178, 220-229. 2008.
- [CPC|02] “Understanding Certification Path Construction”. PKI Forum White Paper, Steve Lloyd, September 2002, <http://www.pkiforum.org/resources.html>
- [CRT] “Chinese Remainder Theorem”, http://en.wikipedia.org/wiki/Chinese_remainder_theorem
- [Diffie-Hellman|76] W. Diffie and M. Hellman. “New directions in Cryptography”. IEEE Trans. On Information Theory, IT-22(6) pages 644-654. November 1976.
- [Dutta|05] R. Dutta and R. Barua. “Overview of Key Agreement Protocols”. Cryptology ePrint Archive, Report 2005/289.
- [Eddie|05] Eddie M. Ng. “Security Models and Proofs for Key Establishment Protocols”. Master’s thesis, University of

Waterloo, 2005.

- [Finke|09] Thomas Finke, Max Gebhardt and Werner Schindler. “A New Side-Channel Attack on RSA Prime Generation”. In CHES 2009, Volume 5747/2009, Springer Berlin/Heidelberg, pages 141-155.
- [Giampaolo|03] G. Giampaolo, F. Masacc F and L. C. Paulson. “Verifying the SET Registration Protocols”, IEEE Journal on Selected Areas in Communications, 21(1), pages 77-87, 2003.
- [Gong|93] Li Gong. “Lower bounds on messages and rounds for network authentication protocols”. In ACM Conference on Computer and Communications Security, pages 26-37, 1993.
- [Gong-Lomas|93] L. Gong, M.A. Lomas, R.M. Needham and J.H. Saltzer. “Protecting poorly chosen secrets from guessing attacks”. IEEE Journal on Selected Areas in Communications 11 (1993) (5), pages 648-656.
- [Gorantla|08] M. Choudary Gorantla, Colin Boyd and Juan Nieto. “ID-based one-pass authenticated key establishment”. ACM International Conference Proceeding Series; Vol. 328, ISBN ~ ISSN:1445-1336 , 978-1-920682-62-0, 2008
- [Guo|09] Hua Guo, Yi Mu, Xiyong Zhang and Zhoujun Li. “Server Controlled Identity-Based Authenticated Key Exchange”. Provable Security, Springer Berlin/Heidelberg, Volume

5848/2009, ISSN 0302-9743 (Print), pages 214-229, Nov 2009.

- [Haidar|09] Ali Nasrat Haidar and Ali E. Abdallah. “Formal Modelling of PKI Based Authentication”. Electronic Notes in Theoretical Computer Science 235, pages 55-70, 2009.
- [Hansen|10] K Hansen, T Larsen and K Olsen. “On the Efficiency of Fast RSA Variants in Modern Mobile Phones”. In International Journal of Computer Science and Information Security, Vol. 6, No. 3, Arxiv preprint arXiv:1001.2249, 2010 – arxiv.org.
- [Heninger|09] Nadia Heninger and Hovav Shacham. “Reconstructing RSA Private Keys from Random Key Bits”. Advances in Cryptology-CRYPTO 2009, Springer Berlin/Heidelberg, Volume 5677/2009, ISBN 978-3-642-03355-1, pages 1-17, 2009.
- [Hölbl|09] Marko Hölbl, Tatjana Welzer and Boštjan Brumen. “Comparative Study of Tripartite Identity-Based Authenticated Key Agreement Protocols”. Informatica 33 (2009), pages 347-355
- [IEEEa] IEEE-P1363.2. “Password based Public Key Cryptography”.
<http://grouper.ieee.org/groups/1363/passwdPK/index.html>
- [Ion|10] Mihaela Ion and Regis Saint-Paul. “Business Impact of Online Institutional Recommendation”. Digital Business,

- Volume 21, ISBN 978-3-642-11531-8, Springer Berlin/Heidelberg, pages 45-52. 2010.
- [ISO08] International Organization for Standardization, Genève, Switzerland. ISO/IEC 11770-3:2008, Information technology Security techniques Key management; Part 3: Mechanisms using asymmetric techniques, 2nd edition, 2008.
- [ISO96] International Organization for Standardization. Genève, Switzerland. ISO/IEC 11770-2:1996. “Key management, Part 2: Mechanisms using symmetric techniques”. 1996.
- [Janson93] Philippe Janson and Gene Tsudik. “Secure and minimal protocols for authenticated key distribution”. Technical report, 1993.
- [Jerschow09] Yves Igor Jerschow, Björn Scheuermann and Martin Mauve. “Counter-Flooding: DoS Protection for Public Key Handshakes in LANs”. ICNS, 2009 Fifth International Conference on Networking and Services, pages 376-382, 2009.
- [Jiang05] S. Jiang and G. Gong. “Password based key exchange with mutual authentication”. In the proceedings of SAC’04, LNCS, vol. 3357, 2005, pages 267-279
- [Jochemsz07] E. Jochemsz. “Cryptanalysis of RSA variants using small roots of polynomials”. PhD Thesis, Technische Universiteit Eindhoven 2007.
- [Joux03] A. Joux. “A one round protocol for tripartite Diffie-

- Hellman”. In the Proceedings of ANTS IV, Journal of Cryptology 17 (2003) (4), pages 263-276.
- [Joux|07] A. Joux, D. Naccache and E. Thom'e. “When e-th roots become easier than factoring”. In ASIACRYPT 2007, volume 4833 of Lecture Notes in Computer Science, pages 13-28.
- [Juang|04] W. S. Juang. “Efficient Three-Party Key Exchange using Smart Cards”. IEEE Transactions on Consumer Electronics 50(2), 619-624, 2004.
- [Jun|06] Z. Jun, Z. Yu, M. Fanyuan, G. Dawu and B. Yingcai. “An extension of secure group communication using key graph”. Information Sciences 176 (20), pages 3060-3078, 2006.
- [Katz|01] J. Katz, R. Ostrovsky and M. Yung. “Efficient password-authenticated key exchange using human-memorable passwords”. In the proceedings of Eurocrypt'01, LNCS, vol. 2045, 2001, pages 475-494.
- [Katz|05] J. Katz and J. Shin. “Modelling insider attacks on group key-exchange protocols”. In the proceedings of ACM CCS'05, 2005, pages 180-189.
- [Kim|04] Young-Sin Kim, Eui-Nam Huh, Jun Hwang and Byung-Wook Lee. “An Efficient Key Agreement Protocol for Secure Authentication”. Published in ICCSA Springer Berlin/Heidelberg, Volume 3043/2004, pages 746-754.
- [Kim|09] Hyun-Seok Kim and Jin-Young Choi. “Enhanced

- password-based simple three-party key exchange protocol". CEE 35, pages 107-114, 2009.
- [Kleinjung|10] Thorsten Kleinjung et al. "Factorization of a 768-bit RSA modulus version 1.3". Cryptology ePrint Archive, Report 2010/006, 2010. <http://eprint.iacr.org/2010/006.pdf>. January 24, 2010.
- [Kohl|93] J. Kohl and C. Neumann. "The Kerberos Network Authentication Service (V5)". Internet Engineering Task Force, RFC 1510, 1993.
- [Ku|00] Wei-Chi Ku and Sheng-De Wang. "Cryptanalysis of modified authenticated key agreement protocol". Electronics Letters, Vol. 36, No. 21, Oct, 2000.
- [Kwon|99] T. Kwon, M. Kang, S. Jung and J. Song. "An improvement of the password-based authentication protocol (K1P) on security against replay attacks". IEICE Transactions on Communications E82-B (1999) (7), pages 991-997.
- [Lagarias|82] J. C. Lagarias, "The computational complexity of simultaneous Diophantine approximation problems", Foundations of Computer Science, Annual IEEE Symposium on, pp. 32-39, 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982), 1982.
- [Lee|04] T.F. Lee, T. Hwang and C.L. Lin. "Enhanced three-party encrypted key exchange without server public keys". Computer Security, 23, 571-577. 2004.

- [Lee|09] Tian-Fu Lee, Jenn-Long Liu, Mei-Jiun Sungc, Shiueng-Bien Yangd and Chia-Mei Chen. “Communication-efficient three-party protocols for authentication and key agreement”. *Computers and Mathematics with Applications* 58, pages 641-648. 2009.
- [Lin|00] Tuon-Chang Lin, Chin-Chen Chang and Min-Shiang Hwang. “Security Enhancement for the simple authentication key agreement algorithm”. *Proceedings of the 24th Annual International Computer Software and Application Conference*, pages 113-115, 2000.
- [Lin-Sun|00] Chun-Li Lin, Hung-Min Sun and Tzonelih Hwang. “Three-party encrypted key exchange: attacks and a solution”. *ACM SIGOPS Operating Systems Review* 34 (2000) (4), pages 12-20.
- [Lin-Sun|01] C.L. Lin, H.M. Sun, M. Steiner and T. Hwang. “Three-party encrypted key exchange without server public-keys”. *IEEE Commun. Lett.*, 5, 497-499. 2001.
- [Liu-Zhang|03] S. Liu, F. Zhang and K. Chen. “ID-based tripartite key agreement protocol with pairing”. In the proceedings of *IEEE ISIT 2003, Yokohama, Japan, 2003*, pages 136.
- [Lo-Yeh|09] N.W. Lo and Kuo-Hui Yeh. “Cryptanalysis of two three-party encrypted key exchange protocols”. *Computer Standards & Interfaces*, 31 (2009), pages 1167-1174.

- [Lu|07] R. Lu and Z. Cao. "Simple three-party key exchange protocol". Computer Security, 26, 94-97. 2007.
- [Mathuria|08] A. Mathuria and G. Sriram. "New attacks on ISO key establishment protocols". Cryptology ePrint Archive: Report 2008/336, July 2008.
<http://eprint.iacr.org/2008/336.pdf>
- [Maurer|94] Ueli M. Maurer, Pierre E. Schmid. Proceedings of European Symposium on Research in Computer Security (ESORICS' 94), Brighton, England, Nov. 7-9, 1994.
- [McCullagh|05] N. McCullagh and P. Barreto. "A new two-party identity-based authenticated key agreement". In the proceedings of CT-RSA'05, LNCS, vol. 3376, 2005, pages 262-274.
- [Menezes|96] Alfred J. Menezes, Paul C. Van Orschot and Scott A. Vanstone. "Handbook of Applied Cryptography". CRC Press, 1996.
- [Milan|10] J Milan. "Factoring Small to Medium Size Integers: An Experimental Comparison". INRIA, CNRS-00188645, version 3, 29 Jan 2010.
- [Molva|92] Refik Molva, Gene Tsudik, Els van Herreweghen and Stefano Zatti. "KryptoKnight Authentication and Key Distribution System". Proceedings of the 1992 European Symposium on Research in Computer Security – ESORICS, pages 1-16, 1992.
- [Morchon|09] Oscar Garcia-Morchon, Tobias Heer and Klaus Wehrle. "Lightweight key agreement and digital certificates for

- wireless sensor networks”. Proceedings of the 28th ACM, ISBN: 978-1-60558-396-9, pages 326-327 2009.
- [Murdoch-Anderson|10] Steven J. Murdoch and Ross Anderson. “Verified by Visa and MasterCard SecureCode: or, How Not to Design Authentication”. January 27, 2010.
<http://www.cl.cam.ac.uk/~rja14/Papers/fc10vbysecurecode.pdf>
- [Nalla|03] D. Nalla. “ID-based tripartite key agreement with signatures”. Cryptology ePrint Archive, Report 2003/144, available at <http://eprint.iacr.org/2003/144/>.
- [Nalla|03] D. Nalla and K. C. Reddy. “ID-based tripartite authenticated key agreement protocols from pairings”. Cryptology ePrint Archive, Report 2003/004, available at <http://eprint.iacr.org/2003/004/>.
- [Needham|78] R. Needham and M. D. Schroeder. “Using encryption for authentication in large networks of computers”. Communications of the ACM, 1978.
- [Nitaj|09] Abderrahmane Nitaj. “Cryptanalysis of RSA Using the Ratio of the Primes”. Progress in Cryptology – AFRICACRYPT 2009, Springer Berlin/Heidelberg, Volume 5580/2009, ISBN 978-3-642-02383-5, pages 98-115.
- [PKCS1] PKCS #1: “RSA Cryptography Standard”,
<http://www.rsa.com/rsalabs/node.asp?id=2125>
- [PKIX] PKIX. “Public-Key Infrastructure – X.509”
<http://www.ietf.org/dyn/wg/charter/pkix-charter.html>

- [RFC 1321] RFC #1321. “The MD5 Message-Digest Algorithm”.
www.faqs.org/rfcs/rfc1321.html
- [RFC 2510] RFC #2510. “Internet X.509 Public Key Infrastructure Certificate Management Protocols”,
<http://www.ietf.org/rfc/rfc2510.txt>
- [RFC 2560] RFC #2560. “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP”.
<http://www.faqs.org/rfcs/rfc2560.html>
- [RFC 2857] RFC #2857. “The Use of HMAC-RIPEMD-160-96 within ESP and AH”. www.ietf.org/rfc/rfc2857.txt
- [RFC 3280] RFC #3280. “Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile”. <http://www.faqs.org/ftp/rfc/pdf/rfc3280.txt.pdf>
- [RFC 3281] RFC #3281. “An Internet Attribute Certificate Profile for Authorization)”. <http://www.faqs.org/rfcs/rfc3281.html>
- [RFC 3874] RFC #3874. “A 224-bit One-way Hash Function: SHA-224”. <http://tools.ietf.org/pdf/rfc3874.pdf>
- [RFC 4055] RFC #4055. “Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”. <http://tools.ietf.org/pdf/rfc4055.pdf>
- [Riad09] Alaa M. Riad, Alaa R. Shehata, Elminir K. Hamdy, Mohammed H. Abou-Alsouad and Taha R. Ibrahim. “Evaluation of the RC4 as a solution for Converged

- Networks”. Journal of ELECTRICAL ENGINEERING, VOL. 60, NO. 3, 2009, 155-160.
- [Rivest-Shamir-Adleman|78] R. Rivest, A. Shamir and L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. ACM 21, pages 120-126, 1978.
- [Riyami|02] S. Al-Riyami and K. Paterson. “Authenticated three party key agreement protocols from pairings”. Cryptology ePrint Archive, Report 2002/035.
- [Riyami|03] S. Al-Riyami and K.G. Paterson. “Tripartite authenticated key agreement protocols from pairing”. In the proceedings of 9th IMA International Conference on Cryptography and Coding, LNCS, vol. 2898, 2003, pages 332-359.
- [Sachin|10] Majithia Sachin and Dinesh Kumar. “Implementation and Analysis of AES, DES and Triple DES on GSM Network”. IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.1, January 2010.
- [Sakai|00] R. Sakai, K. Ohgishi and M. Kasahara. “Cryptosystems based on pairing”. Symp. On Cryptography and Information Security, Okinawa, Japan, Jan. 26-28, 2000.
- [Sarkar|09] Santanu Sarkar and Subhamoy Maitra. “Improved Partial Key Exposure Attacks on RSA by Guessing a Few Bits of One of the Prime Factors”. In ICISC 2008, Volume 5461/2009, ISBN 978-3-642-00729-3, pages 37-51.

- [Schiller|88] J. Schiller and C. Neuman. "Kerberos: An authentication service for open network systems". In Usenix Winter Conference, pages 191-202, January 1988.
- [Schneier|95] Bruce Schneier. "Applied Cryptography-Protocols, Algorithms and Source Code in C". 2nd edition, John Wiley & Sons, Inc., 1995.
- [Seo-Sweeney|99] Dong Hwi Seo and P. Sweeney. "Simple authenticated key agreement algorithm". Electronics Letters, Vol. 35, No. 13, June, 1999.
- [Shamir|85] Adi Shamir. "Identity-based Cryptosystems and Signature Schemes". In Advances in Cryptology. Crypto 1984. Lecture Notes in Computer Science, Vol. 196. Springer-Verlag 1985 pages 47-53.
- [Shamir-Tromer|03] Adi Shamir and Eran Tromer. "On the Cost of Factoring RSA-1024". RSA CryptoBytes, 2003.
- [Shehri|10] Al-Shehri and Khalid N. "Encryption primitives and their application to stream ciphers design". College of Engineering, Jan 2010 available at <http://repository.ksu.edu.sa/jspui/handle/123456789/8694>.
- [Shim|03] K. Shim. "Efficient one round tripartite authenticated key agreement protocol from the Weil pairing". Electronics Letters 38 (2003), pages 208-209.

- [Shim|03b] K. Shim. "A man-in-the-middle attack on Nalla-Reddy's ID-based tripartite authenticated key agreement protocol". Cryptology ePrint Archive, Report 2003/115.
- [Smart|02] N.P. Smart. "Identity-based authenticated key agreement protocol based on Weil pairing". Electronics Letters 38 2002 (13), pages 630-632.
- [Smith-Tritilanunt|07] J. Smith, S. Tritilanunt, C. Boyd, J. M. Gonz'alez Nieto and E. Foo. "Denial-of-service resistance in key establishment". International Journal of Wireless and Mobile Computing. 2007, pages 59-71.
- [Stebila-Ustaoglu|09] Douglas Stebila and Berkant Ustaoglu. "Towards Denial-of-Service-Resilient Key Agreement Protocols". In the proceedings of 14th Australasian Conf. On Information Security and Privacy (ACISP) 2009, LNCS, volume 5594, pages 389-406. DOI: 10.1007/978-3-642-02620-1_27. Springer, 2009.
- [Steiner|98] J.G. Steiner, C. Newman and J.I. Schiller. "Kerberos: an authentication service for open network systems". In the proceedings of USENIX Winter Conference 1998, pages 191-202.
- [Steiner-Tsudik|95] M. Steiner, G. Tsudik and M. Waidner. "Refinement and extension of encrypted key exchange". ACM SIGOPS Operating Systems Review 29 (1995) (3), pages 22-30.
- [Strangio|06] M. Strangio. "An Optimal Round Two-Party Password-Authenticated Key Agreement Protocol". In The First

International Conference on Availability, Reliability and Security, pages 8, 2006.

- [Studer|08] Ahren Studer, Christina Johns et al. "A Survey to Guide Group Key Protocol Development". ACSA Conference 2008, pages 1063-9527/08, DOI 10.1109/ACSAC.2008.28.
- [Sun-Chen|05] H.M. Sun, B.C. Chen and T. Hwang. "Secure key agreement protocols for three-party against guessing attacks". Journal of System and Software, 75, 63-68. 2005.
- [Sun-Wu-Chen|07] H.-M. Sun, M.-E. Wu and Y.-H. Chen. "Estimating the prime-factors of an RSA modulus and an extension of the wiener attack". In ACNS 2007. LNCS, vol. 4521, pages 116-128. Springer, Heidelberg.
- [Szewczyk|01] Perrig, R. Szewczyk, V. Wen, D. Culler and J. D. Tygar. "SPINS: Security protocols for sensor networks". In Seventh Annual International Conference on Mobile Computing and Networks (MobiCom 2001), pages 189-199, July 2001.
- [Tatebayashi|90] M. Tatebayashi, N. Matsuzaki and D.B. Newman Jr. "Key distribution protocol for digital mobile communications systems". Advances in Cryptology, Crypto'89, Springer, Berlin, 1990, pages 324-333.
- [TCG] Trusted Computing Group.

<http://www.trustedcomputinggroup.org/>

- [Thomas|00] S. Thomas. “SSL and TLS Essentials, Securing the Web”, Wiley, 2000.
- [Tseng|00] Yuh-Min Tseng. “Weakness in simple authenticated key agreement protocol”. Electronics Letters, Vol. 36, No. 1, Jan, 2000.
- [Tso|05] R. Tso, T. Okamoto, T. Takagi and E. Okamoto. “An ID-based Non-Interactive Tripartite Key Agreement Protocol with K-Resilience”. Communications and Computer Networks, pages 38-42 (2005).
- [Tyan|02] Y. Her-Tyan and S. Hung-Min. “Simple Authenticated Key Agreement Protocol Resistant To Password Guessing Attacks”. ACM SIGOPS Operating Systems Review, vol. 36, no. 4, pages 14-22, October 2002
- [Wang|05] Y. Wang. “Efficient identity-based and authenticated key agreement protocol”. Cryptology ePrint Archive, Report 2005/108.
- [Wanga|09] Shengbao Wanga, Zhenfu Cao, Kim-Kwang Raymond Choo and Lihua Wang. “An improved identity-based key agreement protocol and its security proof”. Information Sciences Volume 179, Issue 3, 16 January 2009, pages 307-318.
- [Wang-Wei|09] Baocang Wang, Yongzhuang Wei and Yupu Hu. “Fast public-key encryption scheme based on Chinese remainder theorem”. Higher Education Press, co-

published with Springer-Verlag, ISSN 1673-3460 (Print)
1673-3584 (Online), Volume 4, Number 2 June, 2009.

- [Wen-Lee|05] H.-A. Wen, T.-F. Lee and T. Hwang. "Provably secure three-party password-based authenticated key exchange protocol using Weil pairing". IEE Proceedings-Communications 152 (2005) (2), pages 138-143.
- [Wilson-Menezes|97] Simon Blake-Wilson and Alfred Menezes 1997. "Security Proofs for Entity Authentication and Authenticated Key Transport Protocols Employing Asymmetric Techniques". Proceedings of Security Protocols Workshop. Lecture Notes in Computer Science 1361/1997 pages 137-158.
- [Wilson-Menezes|98] Simon Blake-Wilson and Alfred Menezes 1998. "Authenticated Diffie-Hellman Key Agreement Protocols". Proceedings of Selected Areas in Cryptography - SAC 1998. Lecture Notes in Computer Science 1556/1998 pages 339-361.
- [X.509] ITU-T X.509 Recommendation. "Information Technology - Open Systems Interconnection - The Directory Public Key and Attribute Certificate Frameworks". (Equivalent to ISO/IEC 9594-8, 2000).
- [Yang-Chang|09] Jen-Ho Yang and Chin-Chen Chang. "An efficient three-party authenticated key exchange protocol using elliptic curve cryptography for mobile-commerce environments".

The Journal of Systems and Software 82, pages 1497-1502, 2009.

- [Yoon-Yoo|08] E.-J. Yoon and K.-Y. Yoo, “Improving the novel three-party encrypted key exchange protocol”. Computer Standards & Interfaces 30 (2008) 309-314.
- [Zhang|04] M. Zhang. “New approaches to password authenticated key exchange based on RSA”. In the proceedings of Asiacrypt’04, LNCS, vol. 3329, 2004, pages 230-244.
- [Zhang-Liu-Kim|02] F. Zhang, S. Liu and K. Kim. “ID-based one round authenticated tripartite key agreement protocol with pairing”. Cryptology ePrint Archive, Report 2002/122, available at <http://eprint.iacr.org/2002/122/>.
- [Zimmermann|95] P. R. Zimmermann. “The Official PGP User’s Guide”. MIT Press, Cambridge, MA, USA, 1995.

Appendix A

Mathematical Backgrounds

This appendix presents all the relevant mathematics such as Complexity theory, Number theory, Integer Modulo and Group Theory [Menezes|96].

A.1 Complexity Theory

Complexity theory provides mechanisms for classifying computational problems based on the required resources, e.g., time, space, random bits, number of processors, etc., to solve them.

Definition 1 An algorithm is a well-defined computational procedure that takes a set of variable inputs and produces a plausible output.

Usually, the input variables are represented in binary. The number of bits in the binary representation of a positive integer n is $1 + \lfloor \log_2 n \rfloor$ bits. For simplicity, the size of n will be approximated by $\lfloor \log_2 n \rfloor$.

Definition 2 The *running-time* of an algorithm on a particular input is the number of primitive operations or steps executed.

Definition 3 The worst-case running-time of an algorithm is an upper bound on the running-time for any input size.

Definition 4 The average-case running-time of an algorithm is the average running-time over all inputs of a fixed size, expressed as a function of the input size.

Definition 5 A *polynomial-time algorithm* is an algorithm whose worst-case running time function is of the form $O(n^k)$, where n is the input size and k is a constant. Any algorithm whose running time cannot be so bounded is called an *exponential-time algorithm*.

A.2 Number Theory

Definition 6 *Greatest Common Divisor* $\gcd(a,b)$ is the largest positive integer that divides both a and b where a and b are two integers.

Definition 7 Two integers a and b are said to be *relatively prime* (or *coprime*) if $\gcd(a,b) = 1$.

A.2.1 Euclidean Algorithm

The Euclidean algorithm is an efficient algorithm for calculating gcd of two integers without factorizing them, as shown in Table 1. Let's say, a, b are two integers and $a > b$ then $\gcd(a, 0) = 0$,

$$\gcd(a,b) = \gcd(b, a \bmod b).$$

Table 16: Euclidean algorithm for calculating gcd of two numbers

```

Input: Integer a, b;
While (b ≠ 0) {
    r = a mod b; a = b; b = r }
return (a);
Output: Integer a.
```

Definition 8 An integer $p \geq 2$ is said to be *prime* if its only positive divisors are 1 and p . Else it is *composite*.

Properties of Euler Phi Function:

- (i) If p is a prime then ϕ is defined as, $\phi(p) = p-1$.
- (ii) The Euler Phi Function is multiplicative, i.e., if $\gcd(m,n) = 1$, then $\phi(m,n) = \phi(m) \cdot \phi(n)$.

A.3 Integer Modulo

Definition 9 If a, b are two integers, then ‘ a ’ is congruent to ‘ b ’ modulo n if and only if n divides $(a - b)$, denoted by, $a \equiv b \pmod{n}$. Some properties that congruence exhibits are as follows:

- (i) $a \equiv b \pmod{n}$ if and only if a and b leave the same remainder when divided by n .
- (ii) $a \equiv a \pmod{n}$, i.e., reflexive.
- (iii) If $a \equiv b \pmod{n}$ then $b \equiv a \pmod{n}$, i.e., symmetry.
- (iv) If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$, i.e., transitivity.
- (v) If $a \equiv a_1 \pmod{n}$ and $b \equiv b_1 \pmod{n}$, then $a+b \equiv a_1+b_1 \pmod{n}$ and $a \cdot b \equiv a_1 \cdot b_1 \pmod{n}$.

Definition 10 Let Z_n denotes a set of n positive integers (0 to $n-1$), then *modular multiplicative inverse* of $x \in Z_n \pmod{n}$ is an integer $y \in Z_n$ such that $xy \equiv 1 \pmod{n}$, if y exists then it is unique denoted by x^{-1} .

A.4 Group Theory

Definition 11 A *group* $(G; *)$ consists of a set G with a binary operation $*$ on G satisfying the following four axioms.

- (i) The group is closed. That is, $a*b \in G$ for all $a, b \in G$.

- (ii) The group operation is associative. That is, $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.
- (iii) An identity element $e \in G$ exists, such that $a * e = e * a = a$, for all $a \in G$. This element is often written as 1_G or just 1 .
- (iv) Inverse of each $a \in G$ exists, where $a^{-1} \in G$, such that $a * a^{-1} = a^{-1} * a = 1$.

Furthermore, a group G is *Abelian* (or commutative) if,

- (v) $a * b = b * a$ for all $a, b \in G$.

Definition 12 Group G is *finite* if $|G|$ is finite. The number of elements in a finite group is called its *order*.

Definition 13 A group G is *cyclic* if there is an element $a \in G$ such that for each $b \in G$, there is an integer i such that $b = a^i$. The element a is called a *generator* of G .

Definition 14 A *ring* $(R, +, \times)$ consists of a set R with binary operations addition and multiplication on R , satisfying the following axioms.

- (i) $(R, +)$ is an Abelian group with identity element denoted by 0 .
- (ii) The operation \times is associative.
- (iii) Multiplicative identity exists, denoted by 1 .
- (iv) Operation \times is distributive over $+$. That is,
 - $a \times (b+c) = (a \times b) + (a \times c)$ and
 - $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in R$.

The ring is a commutative ring if $a \times b = b \times a$ for all $a, b \in R$.

Definition 15 A *field* is a commutative ring in which all non-zero elements have multiplicative inverses.

Definition 16 Z_n^* is the *multiplicative group* of Z_n if, for every $a \in Z_n$, $\gcd(a, n) = 1$. If n is a prime number then, every $a \in Z_n^*$ such that $1 \leq a \leq n-1$.

Definition 17 The *order* of Z_n^* is the number of elements in Z_n^* . If $a \in Z_n^*$ and order of a is $\phi(n)$ then a is said to be the *generator* of Z_n^* . If Z_n^* has a generator it is said to be cyclic.

Appendix B

Snapshots of the AK-protocol Implementations

The snapshots as shown in, Figures 20, 21 and 22, are from the Java implementations of the AK-protocol on TCP layer. For the simplicity of taking snapshots we are running the Server and the Clients in localhost, however, for the experiments they work in different machines. The output in the screen is to verify the example mentioned in the Section 6.5.2.

```

C:\Windows\system32\cmd.exe - java server.Server3 1500
cd bin > java server.Server3 1500
-----
K      61938743289
X-1    7881879632229573902
N      864608136454559457049
P      389393
Q      262594858
n      2
-----

Server Up , Listening on Port - 1500

[Client INITIATED Socket connection to the Server]
Connection Info { SocketAddr=127.0.0.1,port=49389,localport=1500 }

-- INITIATE CLIENT AUTHENTICATION PROCESS BEGINS on Request FROM Client A --
1. RCD CLIENTA's PUBLIC CERT - C:\clientAPubCert.der
2. REQUESTING PEER CLIENT FOR ITS PUBLIC CERTIFICATE
3. RCD CLIENTB's PUBLIC CERT - C:\clientBPubCert.der
4. SUCCESSFULLY VALIDATED CLIENT CERT's
-- CLIENT A and CLIENT B's SERVER AUTHENTICATION COMPLETED AND SUCCESSFUL --

-- Sending Protocol Message to ClientB - {x-1.n^q, PubA} PubB --
x-1.n^q      326753863534853862509
{x-1.n^q, PubA} PubB
[DATA=3624236233461722317643658250803545936;PEERKEY=5632580245147205390608332358
277385220]

-- Sending Protocol Message to ClientA - {x.n^r, PubB} PubA --
x.n^r      642251862079891166423
{x.n^r, PubB} PubA
[DATA=2448537185616354252124524736201221866;PEERKEY=4527862294991613195978084790
922337000]

SERVER WORK COMPLETED
Closing socket - SocketAddr=127.0.0.1,port=49389,localport=1500

```

Figure 20: Sample 2 Snapshot of Server Module

```

C:\Windows\system32\cmd.exe - java client.ClientA localhost 1500 1600
lei\bin>java client.ClientA localhost 1500 1600
-----
PubKeyA      94729836477
PucKeyA      2428924935109769594914530477100477645
N            864608136454559457049
P            108086391056891903
Q            5534023221128654847
P-Q          5981525981032121372618813159076003841
-----

ClientA Up , Listenning On Port - 1600

MESSAGE TO BE SENT TO(CONNECT;<destIp>;<destPort>)?
CONNECT;localhost;1500

ENTER THE REAL MSG
CONNECT;localhost;1700

INITIATE SERVER AUTHENTICATION
Sending ClientA's Public Certificate - C:/clientAPubCert.der
AUTH_APPROVED RCD from SERVER

-- Protocol Message rcd from Server - <x.m^r, PubB> PubA --
<x.m^r, PubB> PubA
APPROVED[DATA=2448537185616354252124524736201221866;PEERKEY=45278622949916131959
78084790922337000]
-- Decrypted Protocol message using ClientA's PrivateKey --
ProtocolMsg      642251862079891166423
ClientBPubKey    29137283681

MESSAGE TO BE SENT TO(CONNECT;<destIp>;<destPort>)?
CONNECT;localhost;1700

-----
SESSION KEY
437872649104419273489
TimeTakenToCreateSessionKey
[Thu Mar 04 05:40:57 IST 2010 - Thu Mar 04 05:40:57 IST 2010] - Time taken in mi
lliseconds - 5
-----

MESSAGE TO BE SENT TO(CONNECT;<destIp>;<destPort>)?

```

Figure 21: Sample 2 Snapshot of Client A

```

C:\Windows\system32\cmd.exe - java client.ClientB localhost 1600 1700
lei\bin>java client.ClientB localhost 1600 1700
-----
PubKeyB      29137283681
PucKeyB      4153442163589190600456383648658904621
N            864608136454559457049
P            108086391056891903
Q            5534023221128654847
P-Q          5981525981032121372618813159076003841
-----

ClientB Up , Listenning On Port - 1700

AUTHENTICATE REQUEST RCD FROM SERVER
Sending ClientB's Public Certificate - C:/clientBPubCert.der
AUTH_APPROVED RCD from SERVER

-- Protocol Message rcd from Server - <x-1.m^q, PubA>PubB --
<x-1.m^q, PubA>PubB
[DATA=3624236233461722317643658250803545936;PEERKEY=5632580245147205390608332358
277385220]
-- Decrypted Protocol message using ClientB's PrivateKey --
ProtocolMsg      326753863534053862509
ClientAPubKey    94729836477

-----
SESSION KEY
437872649104419273489
TimeTakenToCreateSessionKey
[Thu Mar 04 05:40:56 IST 2010 - Thu Mar 04 05:40:56 IST 2010] - Time taken in mi
lliseconds - 6
-----

```

Figure 22: Sample 2 Snapshot of Client B