

AN ABSTRACT OF THE THESIS OF

Archana Nandakumar Thampi for the degree of Master of Science in Computer Science presented on September 2, 2014.

Title: Efficient, Secure and Covert Channel Capacity Bounded Protocols For Multilevel Security Cross-domain Environments-An Experimental System

Abstract approved: _____

Thinh P. Nguyen

The communication in MLS cross-domain environments faces many challenges. The three most important challenges are efficient key management, privacy preserving and covert channel. We propose an Efficient, Secure and Covert Channel Capacity Bounded Protocol which has three algorithms that addresses these challenges: The Efficient Attribute-based Fine-Grained Authentication (EAFA) algorithm, Anonymous Authentication (A2) algorithm and Limiting Covert Channel Capacity (LC3) algorithm. We implemented a prototype of the ESC3B protocol and measured the performance and efficiency of the system.

©Copyright by Archana Nandakumar Thampi
September 2, 2014
All Rights Reserved

Efficient, Secure and Covert Channel Capacity Bounded Protocols For
Multilevel Security Cross-domain Environments-An Experimental System

by

Archana Nandakumar Thampi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented September 2, 2014
Commencement June 2015

Master of Science thesis of Archana Nandakumar Thampi presented on
September 2, 2014.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Archana Nandakumar Thampi, Author

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. Thinh Nguyen for his extraordinary support and guidance. I also thank my committee members Dr. Bella Bose, Dr. Mike Rosulek and Dr. Clayton Petsche for reviewing my work and serving on my committee.

I thank the members in my group for their technical advice and guidance, and all my friends at Oregon State University for their support.

Finally, I would like to express my special thanks to my husband, my parents, my sister and brother for their encouragement and support.

TABLE OF CONTENTS

	<u>Page</u>
1 Introduction	1
1.1 Identification and significance of problem	1
1.2 Background	1
1.2.1 Service Authentication	2
1.2.2 Preserving Privacy	2
1.2.3 Covert Channel	3
1.3 Proposed solution	3
1.4 Overview	4
2 ESC3B Protocol	5
2.1 EAFA Algorithm	5
2.1.1 Hierarchical Key Structure	6
2.1.2 Exponential Key Generation	9
2.2 Anonymous Authentication (A2) Algorithm	9
2.2.1 Credential Generation (CG)	10
2.2.2 Credential Authorization (CA)	12
2.2.3 Authentication Protocol	13
2.3 Limiting Covert Channel Capacity (LC3) algorithm	17
2.3.1 Existing Protocols for MLS cross-domain communication	17
2.3.2 Data Pump	19
2.3.3 Operation of Pump	19
2.3.4 The Covert Timing Channels	21
2.3.5 Feedback Rate Control	21
2.3.6 Limiting Covert Channel Capacity	23
2.3.7 Pump Algorithm	25
3 Prototype Architecture	27
3.1 High-level Software Architecture	27
3.1.1 EAFA Implementation	27
3.1.2 A2 Implementation	28
3.1.3 LC3 Implementation	29
3.2 Developing Interface to Client	30
3.3 Improvements and Enhancements	32
3.3.1 Dynamically Discovering and Binding to a Web Service	32

TABLE OF CONTENTS (Continued)

	<u>Page</u>
3.3.2 Dynamic Argument Passing	33
3.3.3 Integrating EAFA, A2 and LC3	34
4 Performance Evaluation	36
4.1 EAFA Testing and Performance Evaluation	36
4.2 A2 Testing and Performance Evaluation	37
4.2.1 Authentication Delay vs. Number of Requests	37
4.2.2 Authentication Delay vs. Number of Clients	38
4.3 LC3 Testing and Performance Evaluation	39
4.3.1 Acknowledgement Delay	39
4.3.2 Throughput	41
4.4 Integrated System Performance	42
4.4.1 Acknowledgement Delay	42
4.4.2 Bit Error Probability	44
4.5 ESC3B Prototype Performance Evaluation	46
4.5.1 Service Delay	47
4.5.2 Average Network Throughput	48
5 Conclusion and Future research	49
Bibliography	50

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1 Fine-Grained Authentication	5
2.2 Hierarchical Key Structure	8
2.3 Credential Generation	10
2.4 Credential Authorization	12
2.5 A2 Prototype for Cross-domain communication	14
2.6 Existing Protocols for MLS cross-domain communication	17
2.7 Message Flow in Data Pump	20
3.1 A2 Algorithm - Classes and Dependencies	29
3.2 Data Pump	30
3.3 User Interface for Web-based Cross-domain Data Sharing	31
3.4 Dynamic Binding for Web-based Cross-domain Data Sharing	33
3.5 Data sharing across domains	35
4.1 Key Generation Time	37
4.2 Authentication Delay vs. Number of Requests	38
4.3 Authentication Delay Vs. Number of Clients	39
4.4 ACK Delay Vs. Acknowledgements without covert transmission	40
4.5 ACK Delay Vs. Acknowledgements with covert transmission	41
4.6 Throughput Vs. Window Size	42
4.7 ACK delay Vs. Acknowledgements Without Covert Timing Channel	43
4.8 ACK Delay Vs. Acknowledgements with covert timing channel	44
4.9 Bit Error Probability	45
4.10 Bits Correctly Interpreted	45

LIST OF FIGURES (Continued)

<u>Figure</u>	<u>Page</u>
4.11 Covert channel capacity Vs. high ack delay	46
4.12 Average Delay Time of Cross-domain Data Sharing vs. File Size	48
4.13 Average Throughput vs. File Size	48

LIST OF TABLES

<u>Table</u>	<u>Page</u>
3.1 EAFA API List	28
4.1 System Configuration	46
4.2 File Types and Sizes	47

Chapter 1: Introduction

1.1 Identification and significance of problem

With the proliferation of security attacks targeted on network traffic, enterprises and organizations are following different security policies and classifications. This has made data transfer across multi-level security cross-domain environments more challenging. One of the solutions for MLS environments is using message filters [1], [8] in which a cross-domain guard (CDG) examines the data traversing between the networks. The CDG will only allow the messages that follow certain security policies to pass through it.

Although CDG limits the sensitive information leak between domains, it does not address more important issues like cross-domain authentication and identity protection. Also there is no efficient mechanism to ensure authentication and key management in these systems. These are some of the main challenges in designing cross-domain protocols for MLS cross-domain environments.

1.2 Background

There are three main challenges faced by the MLS cross-domain environments. They are: service authentication, preserving privacy and covert channel.

1.2.1 Service Authentication

Currently the service authentication in MLS cross domain environments uses different keys for different resource authentication. So, large storage space is needed for storing the keys and key management is expensive. For instance, if a user is compromised, all the keys have to be updated, including the legitimate user's, in order to prevent the compromised user from accessing the resources. This causes communication overhead and delays. Hence the current service authentication system is inefficient.

1.2.2 Preserving Privacy

To access the services on another domain, the current approaches require the identity of the client to be revealed to the service provider. The current cross-domain service invocation protocols uses SAML protocols [9] to authenticate the service in a different domain. In MLS cross-domain environments, where the high clearance user does not want to reveal the identity to the service provider in low clearance domain, we need an authentication technique where the low clearance service provider can authenticate the high clearance user without knowing the user's identity. Also, the service provider must be authenticated to the user to make sure that they are interacting with a genuine service provider.

1.2.3 Covert Channel

The information traffic from high security domain to low security domain causes sensitive information to leak. According to the Bell-LaPadula model [3], such information transfers should not be allowed through the covert channel. But this means that the low security domain will not receive any acknowledgement for the data it transferred to the high security domain. Thus, the low security domain has no way to know if the data transfer was successful. Many solutions were suggested to address this issue. One of the notable solutions is store-and-forward (SAF) protocol which uses a gateway between the low and high security domains.

1.3 Proposed solution

The proposed solution is efficient, secure and covert channel capacity bounded (ESC3B) protocols that addresses the three challenges mentioned above for MLS cross-domain environments. The ESC3B protocol consists of three algorithms that help to address these challenges. The first challenge, service authentication and key management, is addressed by the Efficient Attribute-based Fine-Grained Authentication (EAFA) algorithm. The second challenge, privacy preservation is solved by Anonymous Authentication (A2) algorithm. And the third challenge, covert channel information transfer is addressed by the Limiting Covert Channel Capacity (LC3) algorithm.

1.4 Overview

In this thesis, we describe and implement the ESC3B protocol. We describe each algorithm in detail, implement the components and analyze the performance of the algorithms individually and as an integrated system. We also develop a user interface to test the ESC3B prototype.

Chapter 2 describes the background of each algorithm in ESC3B. In Chapter 3, we describe the system architecture, design and implementation details of the EAFA, A2 and LC3 algorithms. We also describe the enhancements we made to improve the prototype. In Chapter 4, we explain the unit testing and performance evaluation details of individual algorithms and the integrated system. Finally, in Chapter 5 we present the conclusion and future work.

Chapter 2: ESC3B Protocol

2.1 EAFA Algorithm

In every network domain, there are multiple resources like database, storage, tactical information, etc. These are locked to prevent illegal access by unauthorized users. Each of these resource is assigned a unique accessing key and only the user with appropriate authentication key is granted access to the corresponding attribute. The proposed scheme follows the RSA cryptosystem [2]. But, for the encryption/decryption mechanism, we use a hierarchical structured keys. In EAFA algorithm, each user needs to store only one key corresponding to the user's clearance level. This reduces the key storage space at the user significantly and simplifies the key management at the service provider side.

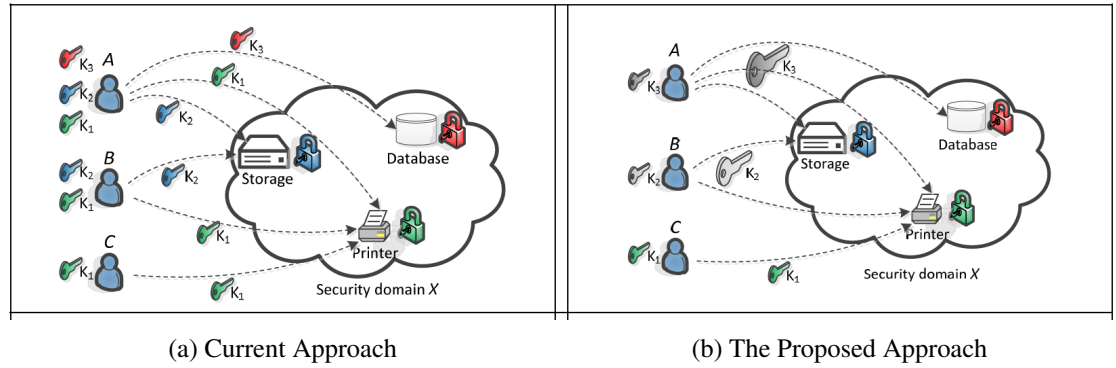


Figure 2.1: Fine-Grained Authentication

Let us assume that the lock and unlock keys used for authenticating a network resource are not necessarily identical. If O is a network resource locked by a key K_l and

unlocked by K_u , where K_l and K_u are not necessarily identical, mathematically the lock and unlock operations are represented as follows:

$$\begin{aligned} \text{Lock} : C &= L_{K_l}(O) = O^{K_l} \pmod{N} \\ \text{Unlock} : O &= U_{K_u}(C) = C^{K_u} = (O^{K_l})^{K_u} = O \pmod{N} \end{aligned}$$

where C is the resource locked by the key K_l . The lock and unlock keys K_l and K_u are chosen such that,

$$\begin{aligned} K_l K_u &\equiv 1 \pmod{\phi(N)} \\ O^{\phi(N)} &\equiv 1 \pmod{N} \\ \gcd(O, N) &= 1 \end{aligned}$$

where $N = p \times q$ for two primes p and q , and $\phi(N)$ is the Euler's totient function (the number of positive integers upto N which are relatively prime to N). i.e, $\phi(N) = (p - 1)(q - 1)$ [5], [12].

2.1.1 Hierarchical Key Structure

In EAFA, a hierarchical key structure is used to provide a multi-level clearance. Higher clearance users are provided powerful keys and lower clearance users are provided weaker keys. The higher clearance user can authenticate more resources using the same key and the lower clearance user can authenticate lesser resources. In current

approaches, the higher clearance user will have to hold all the keys needed to decrypt corresponding resources. This causes key management issues and security risks.

The proposed approach provides a hierarchical key structure so that each user just needs to hold only one unique key to access the data in all security classes. This helps to simplify the key management, reduces key storage space and provide increased key protection. Consider a function $D(N)$ such that,

$$D(N) = c\phi(N) + 1$$

where c is any constant and $c > 0$. Thus,

$$D(N) \equiv 1 \pmod{\phi(N)}$$

Let (d_1, d_2, \dots, d_n) be the set of divisors of $D(N)$ where $d_1 < d_2 < \dots < d_n$. We can group these divisors in a pair-wise form $((d_1, d_n), (d_2, d_{n-1}), \dots, (d_k, d_{n-k+1}))$ where the product of each pair equals $D(N)$. The key pair can be written in a generalized form as d_{n-j}, d_{j+1} . So, if we lock a resource using any key pair d_{n-j}, d_{j+1} , where $d_{j+1} \mid d_k$ (i.e., d_k is divisible by d_{j+1}), then we can authenticate that resource using key d_k since $d_k = \alpha d_{j+1}$, where $\alpha > 0$ and $j < n$. Therefore,

$$d_{n-j}d_k \equiv d_{n-j}(\alpha d_{j+1})$$

The actual key used for authentication can be computed as:

$$K_{actual} = d_k / \alpha = d_{j+1}$$

Thus, if we have three clearance levels A, B and C, by choosing parent key pairs (d_1, d_n) , (d_2, d_{n-1}) and (d_3, d_{n-2}) such that $d_1 \mid d_2$, $d_1 \mid d_3$ and $d_2 \mid d_3$, we can lock all A-clearance attributes using d_{n-2} , all B-clearance attributes using d_{n-1} , and all C-clearance attributes using d_n . Thus, an A-clearance user can authenticate A, B and C-clearance attributes. The B-clearance users can authenticate B and C-clearance attributes, but the C-clearance users can only access C-clearance attributes. C-clearance users will not be able to access A-clearance attributes because d_3 does not divide d_1 , thus K_{actual} will be < 1 .

Figure 2.2 illustrates the hierarchical key structure. When distributing the keys to users, the service provider keeps N , $\phi(N)$ and K_l as secret.

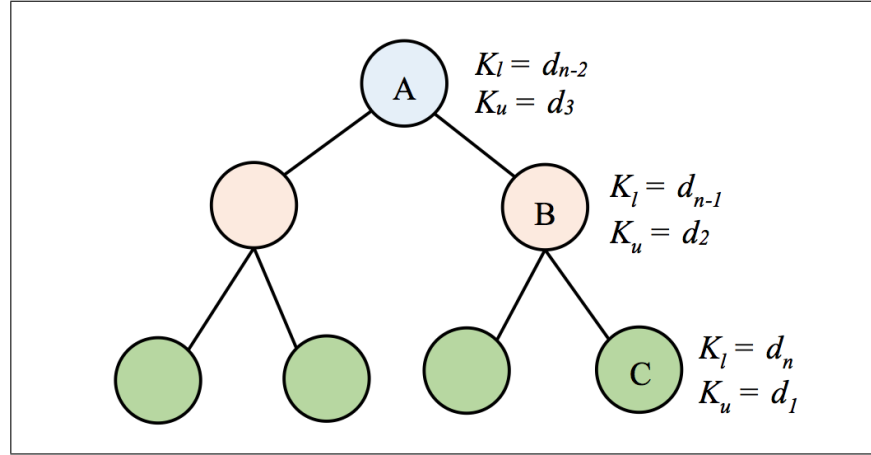


Figure 2.2: Hierarchical Key Structure

2.1.2 Exponential Key Generation

Although the above method allows us to generate keys for authenticating resources, it might take a long time to find the divisors of $D(N)$. We can use exponential operation to generate unlocking keys corresponding to the locking key K_l where the original unlocking key is K_u . $\{K_l, K_u\}$ is called the parent key pair and all the other keys derived from it are called children key pairs.

To generate children keys using exponential function, the N need not be a product of two prime numbers. For any two arbitrary integers X and Y satisfying the equation, $Y - X = \phi(\phi(N))$, the children key pairs $(K_u)^X, (K_l)^Y$ are derived from the parent key pair $\{K_l, K_u\}$ where $K_l K_u \equiv 1 \pmod{\phi(N)}$.

2.2 Anonymous Authentication (A2) Algorithm

When a user in one domain tries to access resources on another domain, for each access we need to perform authentication across domains. By using the proposed A2 algorithm, we can perform the authentication operation by hiding the user's identity.

Also, from the aspect of a distributed application where the subprograms (residing in different domains) want access to some resources in another domain(s), by using the proposed scheme, the subprograms do not need to authenticate every time they access the designated resources across the domains. This will significantly reduce the time delay, performance and chances for covert communications between the domains. The A2 algorithm mainly has two key functions, Credential Generation (CG) and Credential Authorization (CA).

2.2.1 Credential Generation (CG)

The user first generates credentials corresponding to the service he wants. These credentials are later authorized by the service provider in the Credential Authorization step. Once authorized, the user uses them to authenticate to the services. Figure 2.3 shows the steps in the Credential Generation process.

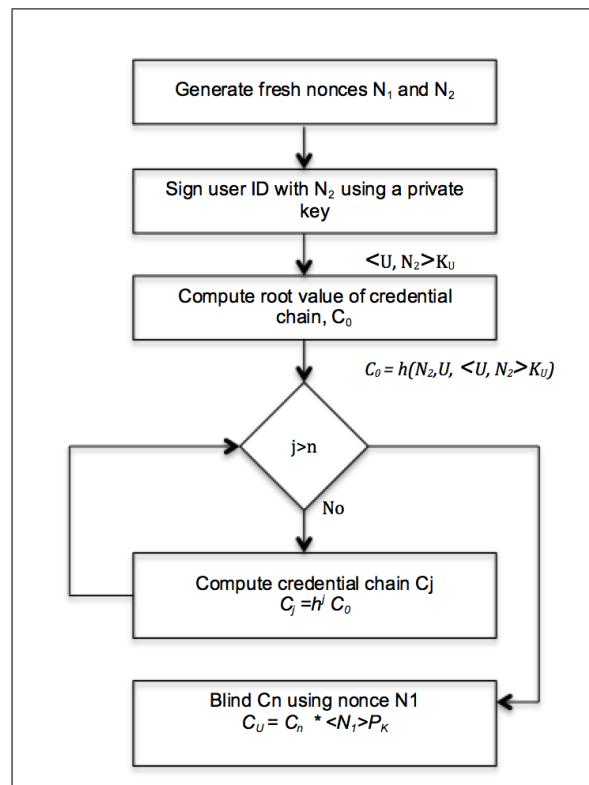


Figure 2.3: Credential Generation

The credential generation procedure consists of the following steps:

- Nonce Generation

The first step of Credential Generation is, generating two fresh nonces N_1 and

N_2 . The purpose of using fresh nonces each time is to prevent the adversaries from using the compromised information of the previous sessions to perform authentication and access to the resources. Also, the fresh nonce will play as the blind factor that prevents the service provider knowing about the signed message. The user then signs a nonce N_2 using its private key, K_U and obtain a message $\langle U, N_2 \rangle_{K_U}$, where U is the identity of the user.

- Computing Root of Credential Chain (C_0)

The user then computes the root value of the credential chain C_0 by using a *one way hashing function* such as SHA-256.

$$C_0 = h(N_2, U, \langle U, N_2 \rangle_{K_U})$$

where $h(.)$ is a secure one-way hash function and K_U is the user private key.

- Computing Credential Chain Values

Now, the credential chain C_j is computed as,

$$C_j = h^j(C_0)$$

where $j \leq n$ and $h^j(.)$ denotes the hash function applied j times on the C_0 . i.e.,

$$h^j(C_0) = h(h(...h(C_0)))_{j \text{ times}}$$

- Computing Blind Credential Chain

Finally, the user computes the blind credential chain as,

$$C_U = \langle N_1 \rangle_{P_K} \times C_n$$

where P_K is the public key of the service provider. Typically, this public key is pre-distributed to the subscribed users.

2.2.2 Credential Authorization (CA)

The user and service provider must authenticate each other first, in order to access the services or resources. To authorize the service, the user will send out the blinded credential chain to the service provider. The credential authorization process is illustrated in Figure 2.4. It consists of two steps:

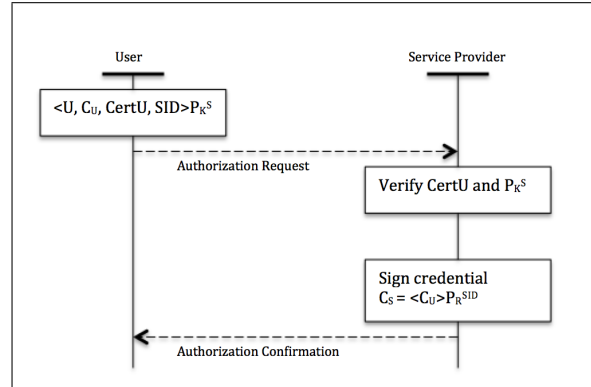


Figure 2.4: Credential Authorization

- Sending Authorization Request

The blinded credential chain C_U is sent to the service provider along with the

user identity ID_U , the service certificate Cert (which binds the user identity ID_U with its public key P_K^U , signed by the private key of the service provider P_R^S), and the identity of the request service. Once the service provider receives the authorization request, it verifies with its public key.

- Receiving Confirmation

After the authorization request is verified, the service provider signs the blinded credential chain C_U as $C_S = \langle C_U \rangle_{P_R^{SID}}$ and sends it back to the user as an authorization confirmation. Upon receiving the authorization confirmation from the service provider, the user decrypts and verifies ID_U and ID_S (the service provider's identity). It then eliminates the blinding factor to obtain the signed credential chain. i.e.,

$$\begin{aligned} (C_S)/N_1 &= \langle \langle N_1 \rangle_{P_K^{SID}} \rangle / N_1 \\ &= N_1 \langle C_n \rangle_{P_R^{SID}} / N_1 \\ &= \langle C_n \rangle_{P_R^{SID}} \end{aligned}$$

Although only the n^{th} credential is signed, the rest of the hash chain values C_0 to C_{n-1} is automatically authorized due to the one way nature of hash function.

2.2.3 Authentication Protocol

We assume that there are two security domains, Low and High. The user/client is in the high security domain and the service provider is in the low security domain. The

high domain contains three components- user/client, service proxy and identity provider (IdP). Similarly the low domain contains the service provider, client proxy and the identity provider proxy (IdPP). The Figure 2.5 illustrates the communication between different components in the high and low domain. The function of each of these components

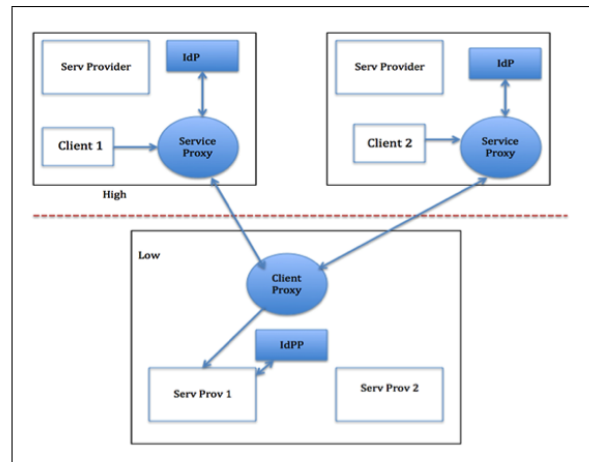


Figure 2.5: A2 Prototype for Cross-domain communication

is defined below:

- **Client/User (U):** This is the application that invokes a service by the users. We assume that the user U has authentic credentials corresponding to the requested services.
- **Identity Provider (IdP):** The IdP manages the users' identity, credentials and the corresponding services (within or across domains) that each user is allowed to access.
- **Service Proxy (SP):** The SP is the component that collects the request sent by the user and forwards it to the IdP and CP.

- Service Provider (P): The service provider P denotes the service requested by the user U. It could be a web service, database access, file transfer, etc.
- Client Proxy (CP): The CP accepts the request from another domain and forwards it to the P. It also sends back the reply from the P to U.
- Identity Provider Proxy (IdPP): The IdPP acts as an intermediate identity verification for the corresponding requested services. IdPP doesn't have any information of the users identity and services.

The authentication protocol has two parts, In-domain authentication and Inter-domain authentication.

- In-domain Authentication

The client sends service request to the service proxy (SP) server located in the domain H. The service request is of the form $(SID, \langle ID_U, N_U, C_j, C_n \rangle_{P_{id}P_k^S})$ where SID is service ID, ID_U is user ID, N_U is a fresh nonce, C_j and C_n are credential chain values. The ID_U , N_U , C_j and C_n are signed by the identity provider. When the credential chain is used for the first time, $C_j = C_n$. So, user sends both C_n and its signature $\langle C_n \rangle_{K_U^R}$ for authentication. In this case, the service request will be $(SID, \langle ID_U, N_U, C_n, \langle C_n \rangle_{K_R^U} \rangle_{P_{id}P_k^S})$. Each credential is used only once. Hence, a credential chain of length n can be used to access the service in n sessions.

The service proxy forwards the request to IdP for authentication. IdP uses its private key $P_{id}P_k^S$ to decrypt $\langle ID_U, N_U, C_n, \langle C_n \rangle_{K_R^U} \rangle_{P_{id}P_k^S}$. It then verifies and authenticates the service request.

- Inter-domain Authentication

After IdP authenticates the request, it encrypts the request using the domain key K_H . Each domain will have a unique key. The information returned to SP is $\{K_H, N_I, \langle N_U, C_j, C_n \rangle_{K_H}\}$. The service proxy will send the request $\{K_H, SID, N_I, \langle N_U, C_j, C_n \rangle_{K_H}\}$ to the client proxy. Client proxy receives the request from High domain and routes it to service provider. The service provider then forwards it to IdPP for authentication. IdPP uses its domain key K_L to authenticate the request by verifying K_H . The IdPP verifies whether the service request is the first one in the credential chain by verifying $C_j = C_n$. Upon successful verification, the identity provider stores C_n and C_j . When it receives the next request, the identity provider verifies whether the value $h(C_j)$ matches the current stored value C_{j+1} . If yes, it updates the stored value to C_j .

After IdPP completes the authentication, it responds back to service provider with authenticated request including $\langle N_I, SID, N_U, C_j \rangle_{P_K^S}$. The service provider P computes two secret keys: $K_{UP} = h(C_j, N_P, N_U, 0)$ and $K'_{UP} = h(C_j, N_P, N_U, 1)$. P sends the authenticated service request $(N_P, N_I, \langle N_U, SID \rangle_{K_{UP}})$ to client proxy which forwards it to SP. SP routes it back to user U. The user then computes the two keys K_{UP}, K'_{UP} by itself and decrypts and verifies N_U, C_j and SID . Now, the user can start secured communication with the service provider using the shared keys K_{UP} and K'_{UP} .

2.3 Limiting Covert Channel Capacity (LC3) algorithm

The data transmission from a low security domain to a high security domain will compromise the system security as the adversary can covertly transmit data over these channels. So it is important to allow the data transmission without compromising the security.

2.3.1 Existing Protocols for MLS cross-domain communication

Some of the existing techniques for secure MLS cross-domain data transmissions are:

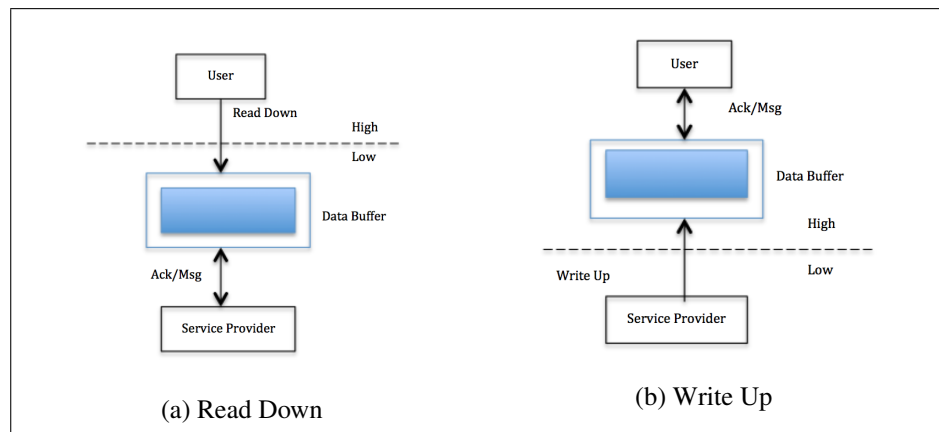


Figure 2.6: Existing Protocols for MLS cross-domain communication

1. The Read-Down Protocol

The read-down protocol for MLS cross-domain communications is illustrated in Figure 2.6a. In this protocol, the low will send its data messages to a data buffer. The high will then read the messages out from the data buffer. Since there is no feedback signal from the high indicating whether it read a message successfully or not, the read-down protocol can eliminate the covert transmission. However,

there exists several issues in data reliability and system efficiency:

- Wasting resources:

Typically in read-down protocol, high will continuously poll the data buffer. In case the Low has lower transmission rate, it will waste high's resources (e.g., CPU time, energy, etc.) as most of the time the buffer is empty.

- Low System Efficiency:

Another way of implementing read-down protocol is to let the high periodically perform read-down after some time interval τ . If τ is too small, it becomes a continuous reading approach, resulting in resource wasting. On the other hand, if τ is too large, the message rates (i.e., system throughput) $1/\tau$ will be small resulting in system inefficiency.

Also, when using this protocol, the low cannot know whether a message has been received successfully by the high or not. If high system gets crashed, then there is no way the low can detect and retransmit the data messages.

2. The Blind Write-up Protocol

An alternate approach for read-down protocol is blind write-up protocol. Figure 2.6b illustrates the blind write-up protocol for cross-domain MLS communication. In this architecture, the data buffer resides in the high domain. The low is allowed to write into the data buffer for sending messages. The high will read the data stored in the data buffer by using ACK/NAK messages. However, the protocol does not allow the data buffer to send feedback messages to the low. This transmission protocol can assure the security (i.e., no covert transmission); however, it

cannot provide reliable communications as the low cannot know which messages have been received successfully.

To allow the high to send feedback to low in the cross-domain MLS environments without enabling a timing covert channel, we will have to limit the feedback message rate without affecting the system efficiency. The Data Pump (DP) explained in the next section helps to limit the covert channel capacity with adjustable parameters for optimal system transmission efficiency.

2.3.2 Data Pump

Data pump is a communication mediator that resides between any two security levels. It can be placed in either low or high security domain. In our prototype, we implemented the data pump in the high domain. The pump is designed in such a way that it only allows ACK/NAK to be sent from high to low and blocks any message flow from high to low. Figure 2.7 shows the structure of the data pump. It mainly has three components: the trusted low buffer (TLB), the trusted high buffer (THB) and the data buffer (BD).

2.3.3 Operation of Pump

The feedback/acknowledgement mechanism using a data pump is explained below. The low process is service provider (lower clearance domain), the high process is the user (higher clearance domain). Data buffer (DB) is a data queue, like a regular FIFO buffer,

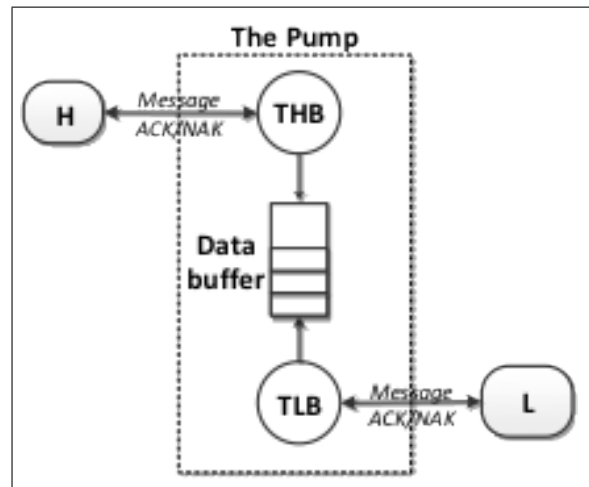


Figure 2.7: Message Flow in Data Pump

which is shared between the THB and TLB.

- The low process (L) sends a message to TLB and waits for a feedback (ACK or NAK message).
- TLB writes the message into data buffer (DB). If the write is successful, it sends an ACK to L after a certain probabilistic delay.
- Once an ACK is received, L removes the packet from its buffer and sends a new message. Otherwise, after a pre-defined timeout or on receiving a NAK, L resends the packet to TLB.
- THB reads the messages from DB and sends it to high process (H). When H successfully receives a data packet from the THB, it sends an ACK back to THB.
- THB on receiving an ACK from H will remove the message from the DB. Otherwise, after a timeout, it retransmits the packet to H.

2.3.4 The Covert Timing Channels

By using the pump, we can isolate the direct communication between the high and low. But, the adversary will still be able to create a covert timing channel by controlling the state of the data buffer as mentioned below:

- Controlling the state of DB : The low adversary can fill up the DB if the high adversary will not remove the messages from DB. Thus, after some time, the DB will become full. This will create a noiseless covert timing channel between high and low.
- Covert communication: In a scenario where the DB is full and the low adversary sends a message to the TLB, the TLB cannot send an ACK back to the low until space opens up on the DB. But that is totally controlled by the high adversary. Let ϵ be the shortest time high takes to remove a message from the DB and allow low to receive an ACK. Based on the time taken to receive the ACK, the low adversary can interpret the symbol corresponding to the ACK time i.e, ϵ and 2ϵ can be interpreted as symbol 0 or 1. Since the DB is kept to be full all the time, the covert channel is noiseless.

2.3.5 Feedback Rate Control

The feedback rate of the data pump can be controlled to make sure that the adversary will not be able to control the channel. Let L_i be the time taken for low to send its i^{th} packet to the TLB and receive an ACK back. If the DB is not full, the packet can be

successfully written into the DB without any delay. Therefore, L_i depends on the fixed communication delay D_l . On the other hand, if DB is full, L_i can be determined by adding a random time R onto D_l . The random time R is the time period TLB has to wait for THB to remove a packet. Mathematically, we can write this as:

$$L_i = \begin{cases} D_l & \text{if } DBO \\ D_l + R & \text{if } DBF \end{cases}$$

where DBO and DBF represent the cases where data buffer has space and data Buffer is full respectively.

Let \bar{H}_m^i represent the average delay of last m ACKs from high when low is transmitting the i^{th} message. So \bar{H}_m^i is the moving average with window size m . The main objective is to achieve high transmission rate while limiting the covert time channel capacity. The optimal system performance is described in the main result given below.

- **Main Result:**

The optimal system performance is achieved if $\bar{L}_i \approx \bar{H}_m^i$ where \bar{L}_i is the expected value of L_i given by,

$$\bar{L}_i = \frac{1}{N} \sum L_i \approx \frac{1}{N} \sum H_m^i$$

- **Proof**

The proof is derived by contradiction.

- If $\bar{L}_i > \bar{H}_m^i$, there is no covert channel, but the system is wasting transmission bandwidth. This results in low transmission rate.
- If $\bar{L}_i < \bar{H}_m^i$, the DB is full and timing covert channel will be exploited.

Thus by contradiction, the optimal performance is attained when $\bar{L}_i \approx \bar{H}_m^i$

2.3.6 Limiting Covert Channel Capacity

The time delay for low to receive an ACK \bar{L}_i depends on the fixed communication delay D_l and delay of an ACK sent by the high to the THB (depending on whether the DB is full or not). The expected waiting time when DB is full is denoted by $\mu\bar{r}$. Let \bar{A} be a random variable that we add to the total delay to control the feedback rate.

To achieve a system that satisfies, $\bar{L}_i \approx \bar{H}_m^i$, we present the below design. We have,

$$P(\bar{L}_i \leq t) = P(\bar{L}_i \leq t \mid DBO)P(DBO) + P(\bar{L}_i \leq t \mid DBF)P(DBF)$$

$$P(\bar{L}_i \leq t \mid DBF)P(DBF) = \sum_k P(\bar{L}_i \leq t \mid DBF, R = r_k)P(R = r_k \mid DBF)$$

If $\{R = r_k\}$ is disjoint event, then

$$P(\bar{L}_i \leq t) = P(\bar{L}_i \leq t \mid DBO)(1 - P(DBF)) + \sum_k P(\bar{L}_i \leq t \mid DBF, R = r_k)P(R = r_k \mid DBF)$$

But if R is defined when DB is full, we can write the above equation as

$$P(L_i \leq t) = (1 - \mu)P(L_i \leq t \mid DBO) + \mu \sum_k P(L_i \leq t \mid DBF, R = r_k)$$

where $\mu = P(DBF)$

Also,

$$P(L_i \leq t \mid DBO) = P(D_l + A \leq t) = P(A \leq t - D_l)$$

$$\text{And, } P(L_i \leq t \mid R = r_k) = P(r_k + D_l + A \leq t) = P(A \leq t - r_k - D_l)$$

By taking derivative on both sides, we get the density function of L_i as,

$$f_{L_i}(t) = (1 - \mu)f_A(t - D_l) + \mu \sum_k p_k f_A(t - r_k - D_l)$$

Therefore the expected waiting time for low is determined as,

$$\bar{L}_i = \int_{-\infty}^{+\infty} t((1 - \mu)f_A(t - D_l) + \mu \sum_k p_k f_A(t - r_k - D_l))dt$$

Let expected value of R be $\bar{r} = \sum_k p_k r_k$. Simplifying the expected wait time, we get

$$\begin{aligned} \bar{L}_i &= (1 - \mu)(\bar{A} + D_l) + \mu \sum_k p_k (\bar{A} + r_k + D_l) \\ &= (1 - \mu)(\bar{A} + D_l) + \mu(\bar{A} + \bar{r} + D_l) \end{aligned}$$

Finally, we get

$$\bar{L}_i = \bar{A} + D_l + \mu \bar{r}$$

If DB is never full, $\mu = 0$. So $\bar{L}_i = D_l + \bar{A}$. On the other hand, if DB is full, $\bar{L}_i = D_l + \bar{A} + \mu \bar{r}$. To obtain the best performance, we will have to choose the value of \bar{A} such that $\bar{L}_i \approx \bar{H}_m^i$ holds. Thus $\bar{A} = H_m^i - D_l - \mu \bar{r}$.

2.3.7 Pump Algorithm

The Pump algorithm allows us to set the value of L_i . Let S_i be the time for low to send a message and receive an ACK back and ε is a small number. D_i is the time that data pump waits to send an ACK to low. The value of D_i is recalculated each time and changes as the moving average H_m^i changes. The relationship between different terms are: $\beta_i = \tau + S_i - H_m^i$, $\tau'_i = \beta'_i + H_m^i - S_i$, $\beta''_i = \beta'_i - S_i$ and a random delay $D_i = L_i - S_i$.

The pump algorithm is described below:

Message sent by low is placed in DB

Read H_m^i

If $S_i \geq H_m^i$ **then**

$\mu := \varepsilon$

Else

$\mu := H_m^i - S_i$

End If

$\zeta = \text{random number of mean } \mu$

If $S_i = D_l$ **OR** $S_i \geq H_m^i$ **then**

$D_i := \zeta$

If $D_l > \tau - S_i$ **then**

$D_i := \tau - S_i$

End If

Else

$\beta'' \leftarrow \text{Random}[0, \beta_i - S_i]$

End If

If $\zeta \leq \beta_i''$ **then**

$D_i := \zeta$

Else

$\zeta' \leftarrow \text{Random}[\tau_i', \tau]$

$D_i := \zeta' - S_i$

End If

Chapter 3: Prototype Architecture

3.1 High-level Software Architecture

We implemented a ESC3B prototype for data sharing from a service provider in low domain to clients in High domain. We implemented the three algorithms: EAFA, A2 and LC3 individually and evaluated the performance. We then integrated the algorithms and tested the whole system. In this prototype, we implemented a web-based application so that the client can send a request to the service provider and retrieve shared data using a web browser. In the following sections we describe the implementation details of each of these algorithms and integration details.

3.1.1 EAFA Implementation

The EAFA algorithm is an important software component to enable secure cross-domain applications in the MLS environments. Using the algorithms explained in section 2.1, we implemented various functionalities of the EAFA. The two classes we implemented are EAFA and Crypt.

The table 3.1 lists the APIs we implemented for EAFA algorithm and its functionalities. These APIs are in the class Crypt and are invoked by the class EAFA. The `main()` function of class EAFA accepts a command line argument which allows the user to specify the bit size of the key. The user can specify 8, 16, 32 etc. as bit size. The hierarchical

API	Functionality
calcParentKeys()	Calculate the parent key pair $\{K_l, K_u\}$
getActualKey()	Calculate the actual key K_{actual} used for authentication
relativePrime()	Find the $\phi(N)$
calcChildKeys()	Calculate the children key pair $\{(K_u)^X, (K_l)^Y\}$
getDivisors()	Get the set of divisors of D to get keys of different levels of clearance.

Table 3.1: EAFA API List

keys obtained using this algorithm is distributed to the users by the service provider. These keys are later used in the A2 stage for authentication.

3.1.2 A2 Implementation

This subsection describes the implementation of A2 algorithm. The software classes, APIs and dependencies are illustrated in Figure 3.1.

The client sends the request to the service proxy using the API `getClientRequest(String request)` in the proxy class. The proxy will redirect this request to identity provider using the API `authenticateClientReq(String request)` in IdP class. IdP internally uses the API `parseAndAuthClientReq(String request)` to parse and authenticate the client request. It checks for the C_n entry in the hash map `cacheClientReq` to determine if this is the first request from the client. The `cacheClientReq` stores the C_n as the key and C_j as the value corresponding to it. Each time the IdP receives a new request from the client, it updates the C_j value.

After authenticating the request, IdP will create the new request with domain key and send it to service proxy. Service proxy will redirect this request to the client proxy using the API `getCrossDomainRequest(String request)`. Client proxy sends it to

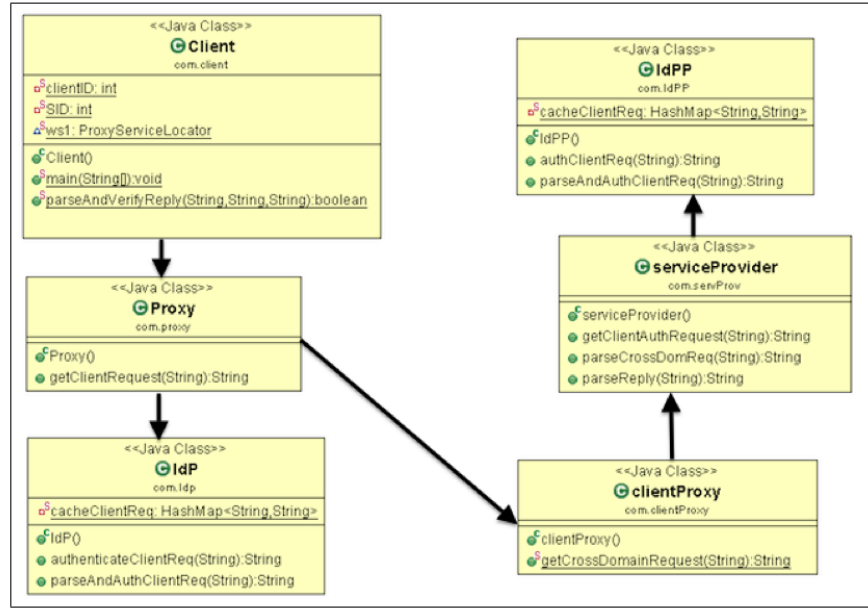


Figure 3.1: A2 Algorithm - Classes and Dependencies

the service provider through the API `getClientAuthRequest(String request)` that is redirected to the IdPP using the API `parseAndAuthClientReq(String request)`. The IdPP authenticates the request similar to IdP. It maintains its own hash Map with the C_n and C_j values.

After authentication, The IdPP returns back the reply to the service provider. Service provider internally calls the API `parseReply(String request)` to parse the IdPP reply and creates the new reply with K_{UP} and K'_{UP} that is returned back to the client.

3.1.3 LC3 Implementation

The main component of the LC3 algorithm is the data pump (DP). DP can lie in the low or high domain. In our prototype, we implemented the DP in the high (User) Domain.

The low, high and DP are web services. Low sends message (chunks of a data file) to the DP, which is inserted into the Data Buffer (DB). The message from DB is sent to the high and removed from the DB when the DP receives an ACK from the high. The software modules of the LC3 algorithm is illustrated in Figure 3.2. The data pump class

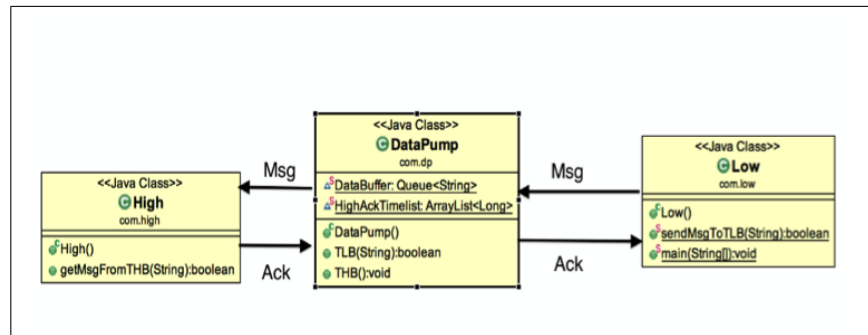


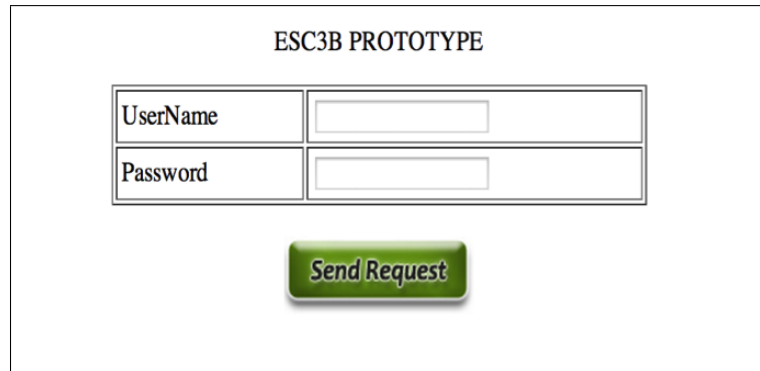
Figure 3.2: Data Pump

has two APIs: TLB() and THB() for receiving from low and sending messages to high respectively. The low sends a message to the TLB() in DP. The message is stored in data buffer (if the DB is not full), which is an FIFO queue. THB() software module will scan the DB and if there is a message, it sends it to the high. The ACK time from high is stored in an array list, i.e., HighAckTimeList. The H_m^i (i.e., the moving average of the last m ACKs) and the ACK delay are calculated in the API findDelay() based on the window size m which can be set to any desired value.

3.2 Developing Interface to Client

To provide an entry interface to the client, we developed a login page. The login page allows the user to enter a user name and password. We assume that different users

in different security clearance levels have different usernames and passwords. Also, we assume that the usernames and passwords are provided beforehand via subscription procedure. Figure 3.3 shows the login page/user interface that accepts the username and



The image shows a web-based login interface titled "ESC3B PROTOTYPE". It contains two input fields: "UserName" and "Password", each with a corresponding text box. Below these fields is a green button labeled "Send Request".

ESC3B PROTOTYPE	
UserName	<input type="text"/>
Password	<input type="password"/>
<input type="button" value="Send Request"/>	

Figure 3.3: User Interface for Web-based Cross-domain Data Sharing

password from the user. In our prototype, we implemented the following functions to handle the entered username/password.

- **Clients Credential Verification:** The password entered is checked for a mapping to EAFA key previously distributed by the service provider to the client.
- **Forwarding Service Request:** Once authenticated, the password entered by the user has a mapping to EAFA key and the client request is initiated and sent to the Service Proxy. If there is no corresponding key, then an error message is displayed on the web browser.

3.3 Improvements and Enhancements

We made many software enhancements to improve our prototype design. Below are some of the changes:

3.3.1 Dynamically Discovering and Binding to a Web Service

In the initial stages of development, we used static client (using stub) to communicate with the web services. We later changed these to dynamic client, which communicates with a web service across domains. Instead of using the stub classes from the web services description language (WSDL), which is generated during the compile time, we modified the code to incorporate the dynamic binding. We used the XML-based Remote Procedure Call (JAX-RPC) standard interfaces like `setTargetEndpointAddress` and `setProperty` to configure the call to the web service. Figure 3.4 shows the dynamic binding flow between client (in high domain) and various web services (in low domain). The software components are communicating with each other (in different machines) via ip address and port number.

The advantages of dynamic client over the static client are:

- Binding of request On-the-fly: Compile time is too early to bind to a web service URL, as we may need to add some configuration or dynamic binding routine later. Dynamic client offers a flexible way to achieve this by allowing the request binding at the run time.
- Simplifying Process Building: With static client, the build process will be more

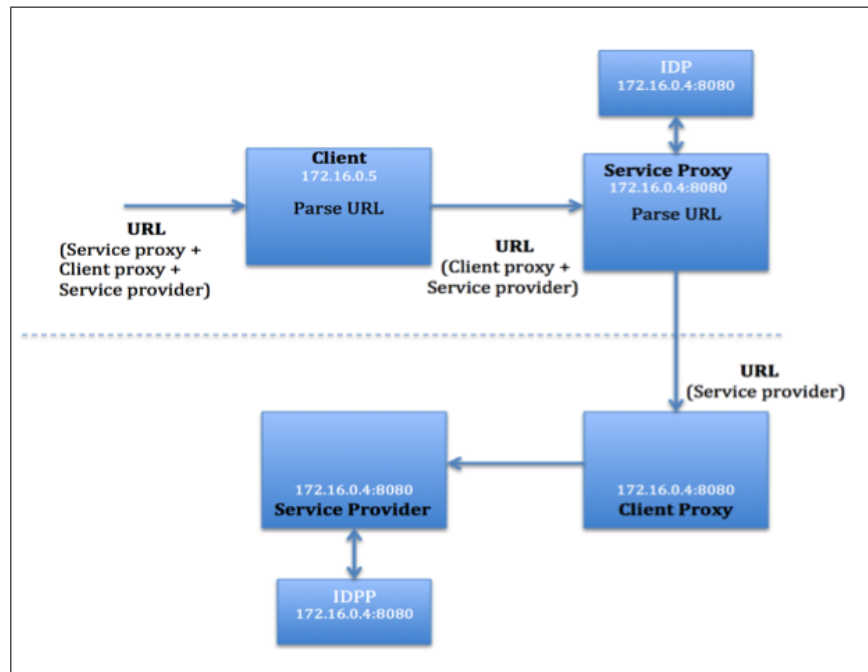


Figure 3.4: Dynamic Binding for Web-based Cross-domain Data Sharing

complex, as we need the web services description language (WSDL) before compiling the client, which may involve deploying the web service beforehand.

- Enabling URL Passing: We can pass the URL of the web services as a command line argument on the run.

3.3.2 Dynamic Argument Passing

We made enhancements to pass the URL of the service requested by the user through the command line of client class `main()` function. This argument is automatically parsed and passed to other software components. Below are the steps on how the URL is parsed.

- The command line argument is passed to the client in the form:

$$http : //ip : port /ServiceProxy/services/Proxy?(client_proxy =$$

$$http : //ip : port /ClientProxy/services/clientProxy?(serv_prov = http : //ip :$$

$$port /ServProv/services/ServProv?cmd = downloadfile))$$
- The above argument is parsed by the client to get the service proxy URL. The rest of the argument ($http : //ip : port /ClientProxy/services/clientProxy?(serv_prov = http : //ip : port /ServProv/services/ServProv?cmd = downloadfile)$) is passed as a parameter to the service proxy call, which is further parsed to obtain the client proxy URL.
- The Service Providers URL ($http : //ip : port /ServProv/services/ServProv?cmd = downloadfile$) is passed as a parameter to the client proxy which is in turn used to make the call to the service provider.
- In our prototype, we assume that the IdP and IdPP URLs are already available (known) to the service proxy and service provider. So these URLs are hardcoded in SP and P respectively.

3.3.3 Integrating EAFA, A2 and LC3

In this section, we explain the integration details of the ESC3B protocols.

In a real time scenario, since the authentication keys will be distributed to the user by the service provider beforehand, we generated the keys using the EAFA algorithm and hardcoded these values in the A2 stage.

We integrated the A2 and LC3 algorithms by adding the data pump component in the high clearance domain along with the client, IdP and service proxy components. We extended the client and service provider by adding some functionalities. In addition to the credentials for authentication, the client request contains parameters specifying service needed (for e.g, file download service and the filename) to the service provider(P). The service request sent by the client does not pass through the DP. The request and the reply flow remains the same. After authentication is successful, the service provider sends back the authentication reply to the client that will in turn compute the corresponding K_{UP} , K'_{UP} keys. The shared keys are used for encrypting/decrypting data transmitted across domains.

After responding to the request, the service proxy will start the file download operation. It opens the file specified by the client and encrypts it using the K_{UP} . To cope with the large file sizes, the service provider performs file processing by dividing the files into smaller data blocks(packets). Each data block will be then encrypted and sent to the client through the DP. The client will decrypt the data using K_{UP} and save it.

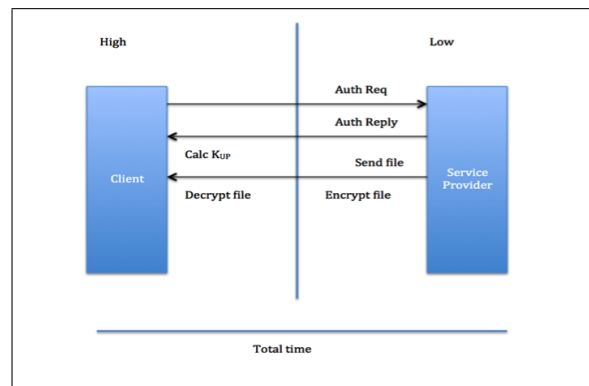


Figure 3.5: Data sharing across domains

Chapter 4: Performance Evaluation

We conducted various tests and experiments to evaluate the performance of the implemented prototype. We first evaluated the performance of each algorithm individually. Then we tested and evaluated the integrated ESC3B prototype.

4.1 EAFA Testing and Performance Evaluation

We evaluated the performance of the EAFA algorithm from different aspects:

- **Correctness:** We tested the correctness of EAFA algorithm to ensure that the algorithm is working as designed. We verified that the encryption key and its corresponding decryption key are working by encrypting/decrypting various files. Additionally, we tested hierarchical keys by checking whether a decryption key of a higher security level can also decrypt the data encrypted by a lower security level encryption keys.
- **Efficiency:** We evaluated the efficiency by testing how long it takes for key generation, data encryption and decryption. Figure 4.1 illustrates the performance comparison of EAFA and RSA algorithms.
- **Security:** We tested the protection level when sharing data across domains. EAFA can ensure that only corresponding security clearance users can decrypt the data

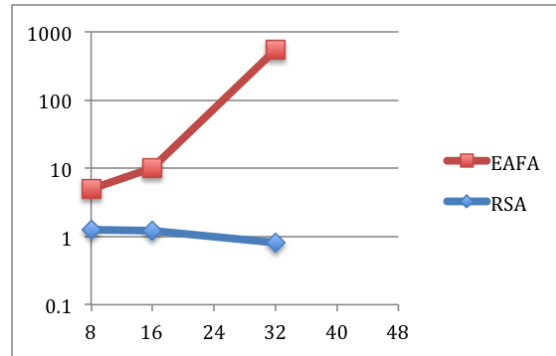


Figure 4.1: Key Generation Time

shared across domains. The lower clearance users learn nothing about the encrypted data of higher security levels using their keys.

4.2 A2 Testing and Performance Evaluation

We tested the A2 algorithm by initiating a service request from user to the service provider and checked the correctness of authentication algorithm at IdP and IdPP. We also checked the K_{UP} and K'_{UP} keys calculated by the user against the keys calculated by the service provider.

Below are the results of the performance evaluation test we conducted on the A2 algorithm.

4.2.1 Authentication Delay vs. Number of Requests

We tested the average delay time against the number of authentications (service requests) with and without using the A2 algorithm. Figure 4.2 illustrates the results. The A2

algorithm requires only one authentication for any number of requests sent by the client using the same C_n , whereas the other (standard algorithm) requires one authentication per request. Also, the average delay time for A2 algorithm is significantly low compared to the algorithm without it.

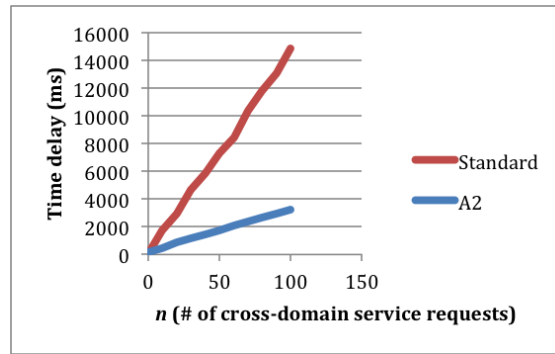


Figure 4.2: Authentication Delay vs. Number of Requests

4.2.2 Authentication Delay vs. Number of Clients

We evaluated the algorithm performance in terms of authentication delay versus the number of users. Figure 4.3 illustrates the delay times versus the number of clients by using A2 algorithm and without it (standard algorithm). The delay time is more for standard algorithm as the verification needs to be done for each request sent by the client. Also, in both the cases, the delay increases as number of clients increases.

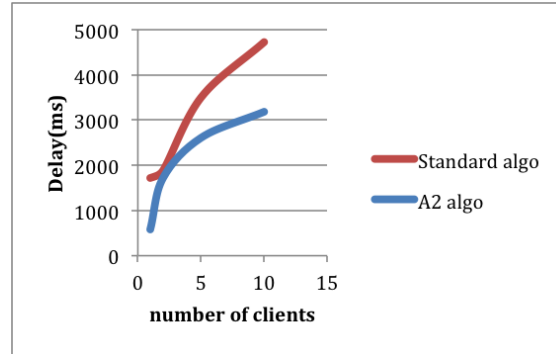


Figure 4.3: Authentication Delay Vs. Number of Clients

4.3 LC3 Testing and Performance Evaluation

The LC3 (data pump) algorithm was tested by sending data packets from service provider (P) to the client (U) through the data pump (DP) and the acknowledgements from U to P. Various performance test results are mentioned below.

4.3.1 Acknowledgement Delay

The acknowledgement delay for low (L_i) using the DP was collected for various window sizes (m) and number of messages (N). We measured the ACK delay with and without a covert timing channel.

- **Without Covert Transmission:** In this scenario, we assume that there is no adversary in the high domain. Thus, the high process will send an ACK immediately after it receives a message from the DP. Figures 4.4a - 4.4d represent the ACK delays Vs. acknowledgements for various window sizes.
- **With Covert Transmission:** In this scenario, we assume that there is an adversary

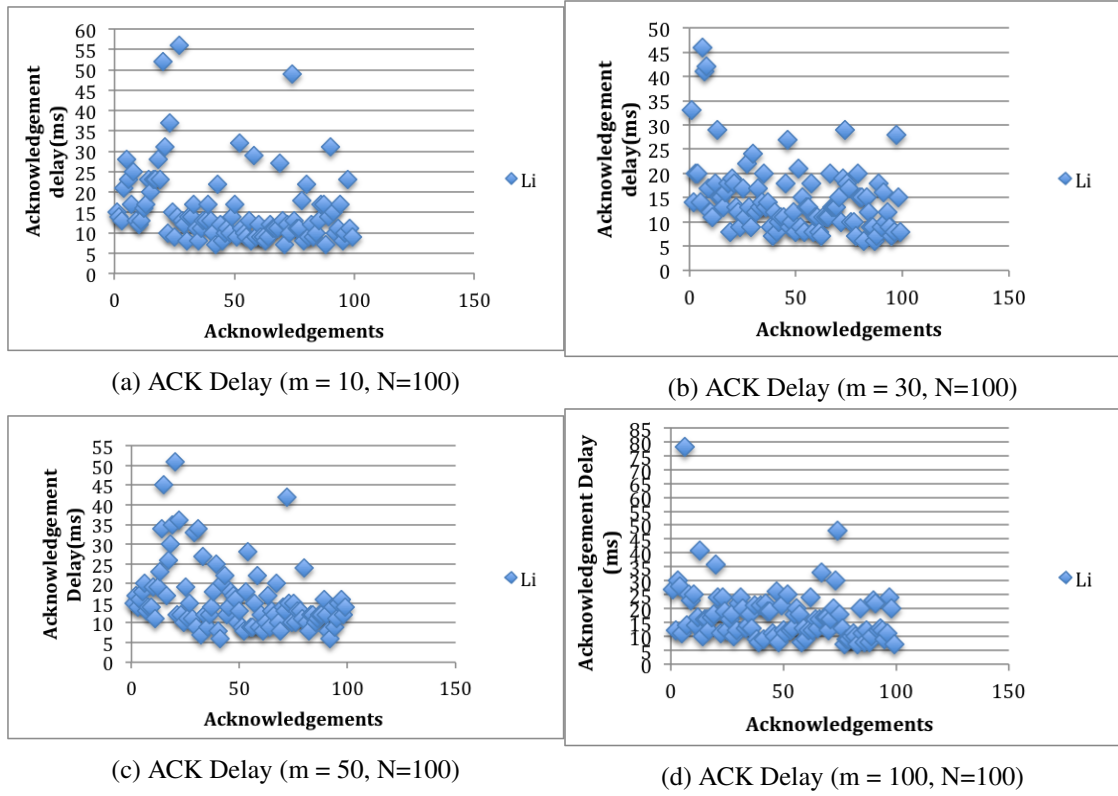


Figure 4.4: ACK Delay Vs. Acknowledgements without covert transmission

in the high domain that controls the high process. The delay time depends on the message the adversary wants to transmit. Figures 4.5a - 4.5c represent the ACK delays for each transmitted data packet assuming that the adversary in the high domain controls the ACK delays for covertly transmitting its information. As expected, the distribution of the ACK delays scattering around two mean values corresponding to time delays made by the adversary in the high.

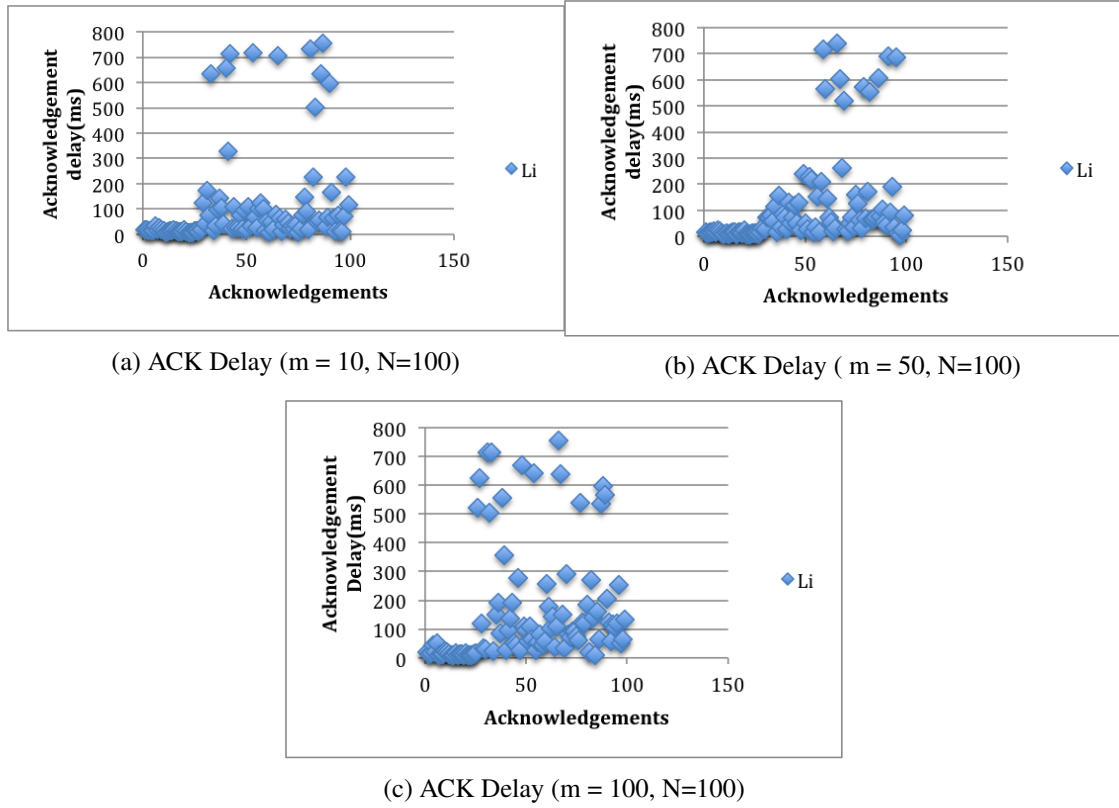


Figure 4.5: ACK Delay Vs. Acknowledgements with covert transmission

4.3.2 Throughput

We compared the throughput of the transmission without and with covert timing channel in Figures 4.6a and 4.6b respectively. We varied the window sizes of the data pump and measured the transmission throughputs. The throughput of the system without the covert timing channel is significantly greater than that of the system with covert timing channel. This is because with covert timing transmission, the adversary in the high domain will control the ACK sending back the DP, thus, slowing down the whole system.

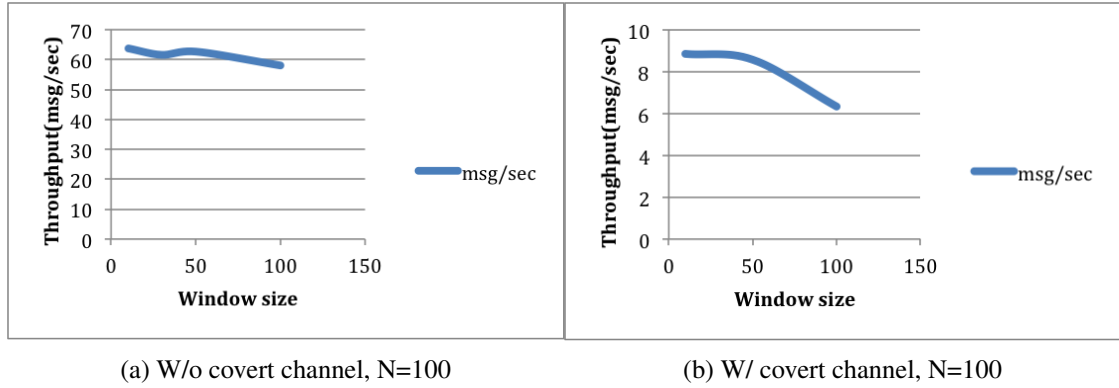


Figure 4.6: Throughput Vs. Window Size

4.4 Integrated System Performance

We evaluated the performance of the integrated system, especially after integrating the data pump with A2 algorithm.

4.4.1 Acknowledgement Delay

- Without Covert Timing Channel :** We evaluated the delays of the ACK messages when there is no covert transmissions. Figures 4.7a - 4.7c show the ACK delay with respect to different window sizes. As expected, when there is no covert transmission, the ACK delays (L_i) for different data packets are almost same as H_m^i . This is because the client sends the ACK right after it receives a data packet successfully.
- With Covert Timing Channel:** We assume that there is an adversary at the client, which wants to covertly transmit information to the other adversary residing in

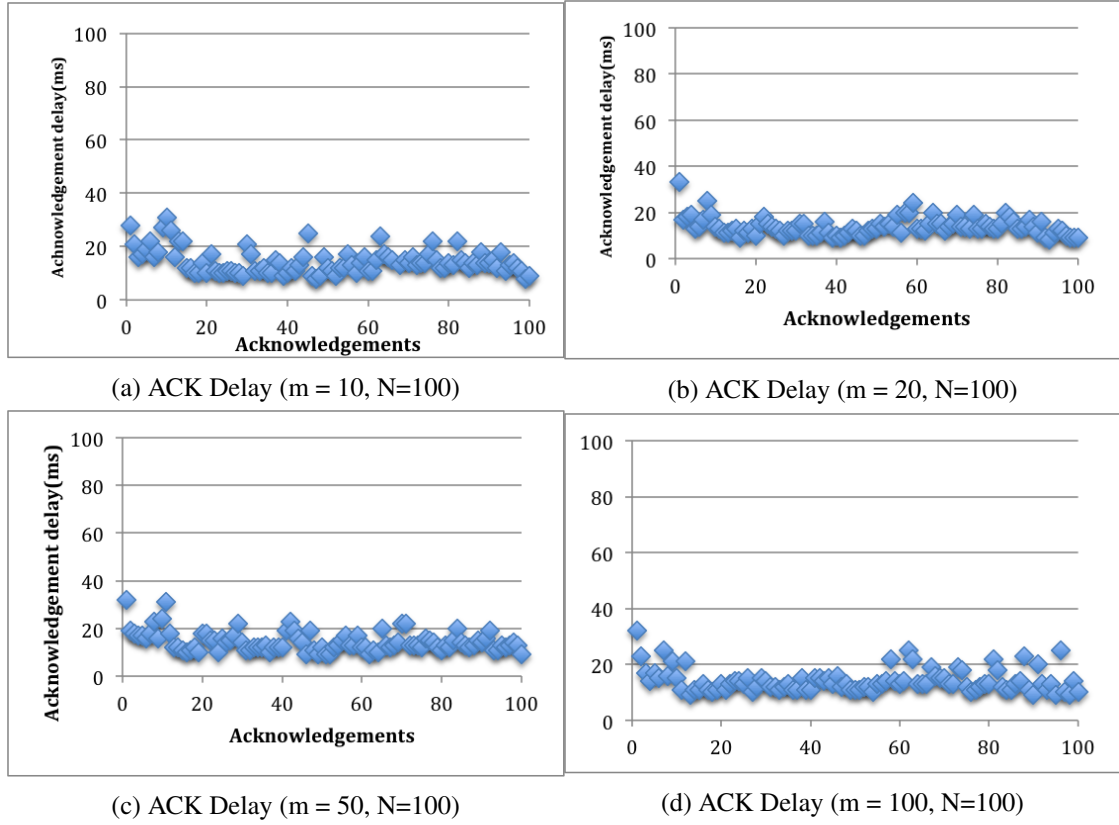


Figure 4.7: ACK delay Vs. Acknowledgements Without Covert Timing Channel

the low domain. To simulate this scenario, we assume that the adversary sends message with bits '0' and '1' following a uniform distribution. To transmit bit '0' the adversary doesn't delay the ACKs and to transmit bit '1', it delays the ACKs. Figures 4.8a - 4.8c illustrate the ACK delays with respect to different window sizes. As expected, the ACKs delays now classified into two clusters corresponding to the transmissions of bits '0' and '1' of the adversary.

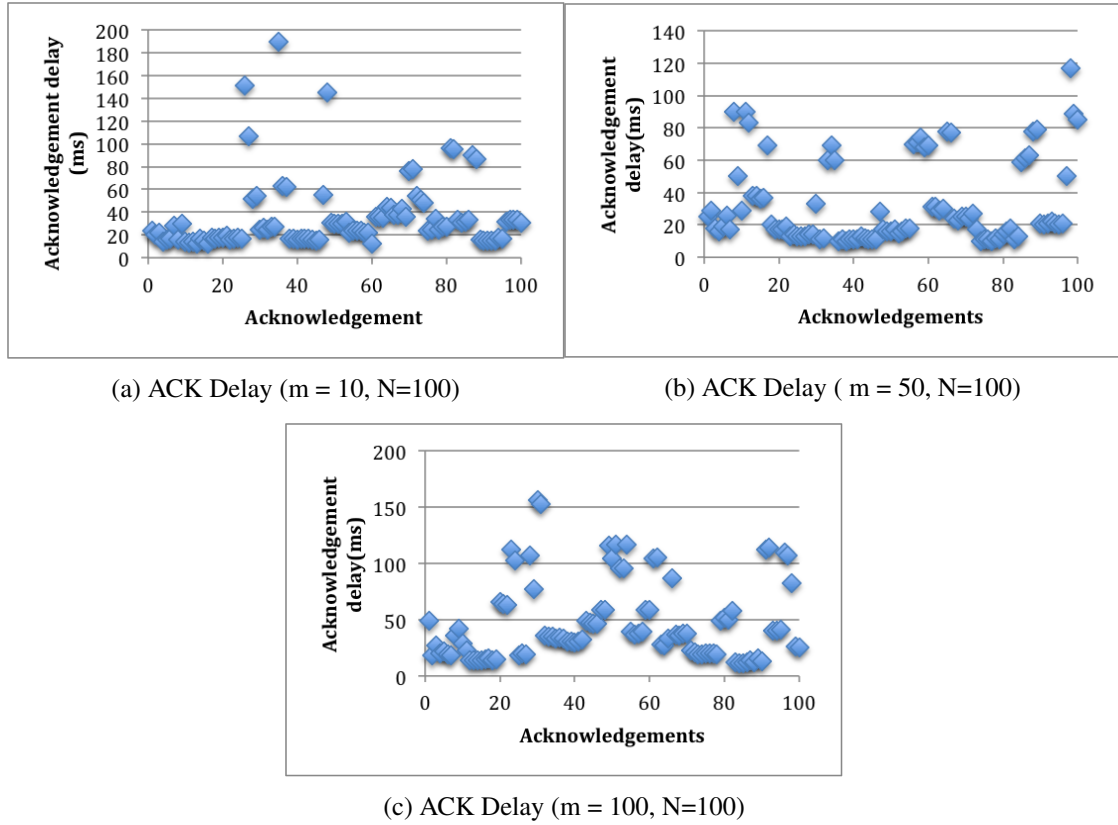
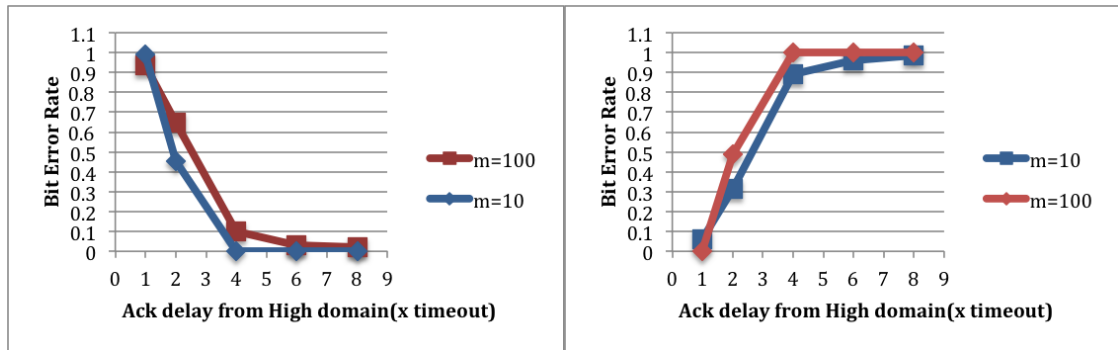


Figure 4.8: ACK Delay Vs. Acknowledgements with covert timing channel

4.4.2 Bit Error Probability

We simulated a covert channel by allowing the data pump to become full and tested the probability of a bit '0' sent from high being interpreted as '1' in low and vice-versa. We calculated the bit error probability and covert channel capacity for various ACK delays from high domain. $P(\frac{y=0}{x=1})$ is the probability of bit '1' sent from high interpreted as '0' in low and $P(\frac{y=1}{x=0})$ is probability of bit '0' sent from high interpreted as '1' in low.

Figures 4.9a and 4.9b show the bit error probabilities for various ack delays from the high. As expected, bit error probability $P(\frac{y=0}{x=1})$ decreases when high ACK delay increases because low domain can distinguish between the different ack delays and $P(\frac{y=1}{x=0})$ increases as high ACK delay increases.

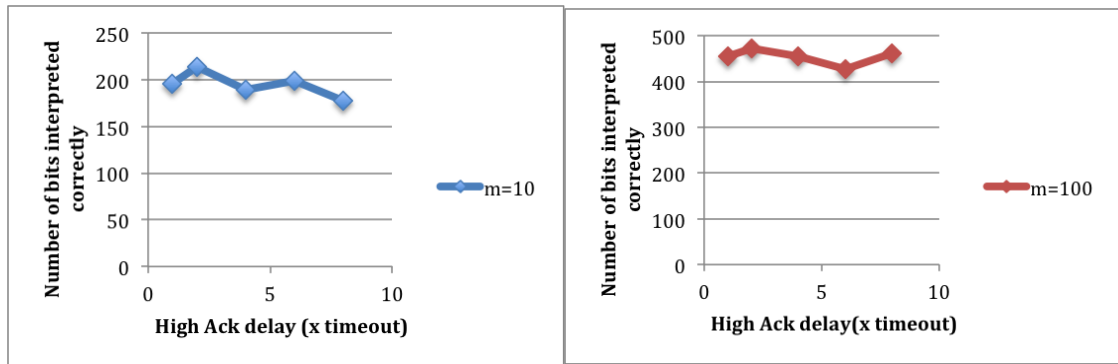


(a) $P(\frac{y=0}{x=1})$ vs. High ack delay

(b) $P(\frac{y=1}{x=0})$ vs. High ack delay

Figure 4.9: Bit Error Probability

Figure 4.10a and 4.10b shows the average number of bits correctly interpreted. Figure



(a) Window size, m=10

(b) Window size, m=100

Figure 4.10: Bits Correctly Interpreted

4.11 shows the covert channel capacity at different high ack delays. As expected, when

window size increases, the covert channel capacity decreases.

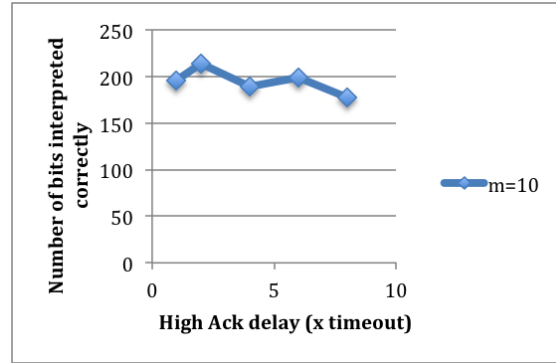


Figure 4.11: Covert channel capacity Vs. high ack delay

4.5 ESC3B Prototype Performance Evaluation

We conducted various tests and experiments to evaluate the performance of the implemented prototype using the user interface. Particularly, we performed data sharing across two domains using different data sizes and formats. We tested the prototype us-

System Configuration	System 1(Service Provider)	System 2(Client)
Processor	2.5 GHz Intel Core i5	2.9 GHz Intel Core i7
Memory	8 GB 1600 MHz DDR3	15.4 GB
Operating System	Mac OS	Linux(Ubuntu)
IP address	172.16.0.4	172.16.0.5

Table 4.1: System Configuration

ing two systems connected via a LAN networks. The client process was deployed on the System 2 and the web services were deployed in System 1, whose configurations are illustrated in table 4.1. We used different file formats and sizes to evaluate the end-to-end

File Type	File size(KB)
PDF	~50, 100, 200, 500, 700
DOCX	~50, 100, 250, 1000
JPEG	~50, 100, 600, 800

Table 4.2: File Types and Sizes

delay as well as data throughput of the implemented prototype. The sizes and formats are illustrated in table 4.2.

To evaluate the performance of ESC3B prototype using the user interface, we measure two important parameters of the cross-domain data sharing as follows.

4.5.1 Service Delay

This metric evaluates the system delay which measures the time interval since a request is sent until whole data file is downloaded. Thus, the service delay can be considered as an end-to-end delay including authentication, data encryption/decryption, data transmission and retransmission of a cross-domain data sharing. Figure 4.12 illustrates the average service delay versus the data file size of different formats. The average service delay of each data format was computed by averaging the service delays of multiple experiments using the same system parameters.

The end-to-end service delay includes the authentication delay, transmission delay and encryption/decryption delay. The average time taken by the client to send an authentication request and receive a reply from the service provider was same for all the file types and file sizes. On the other hand, the time to encrypt a file and send it to client and

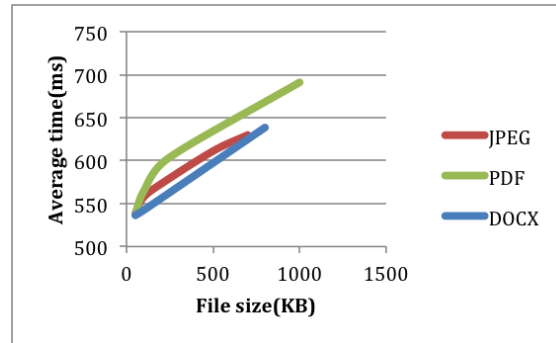


Figure 4.12: Average Delay Time of Cross-domain Data Sharing vs. File Size

decrypt it at the client side varies according to the file size and types. As a result, the average delays increase with the file sizes of the data files.

4.5.2 Average Network Throughput

Figure 4.13 illustrates the average network throughput of different data formats versus the file sizes. As expected, the transmission throughput across domains between the client and service provider is stable regardless of the data sizes and formats.

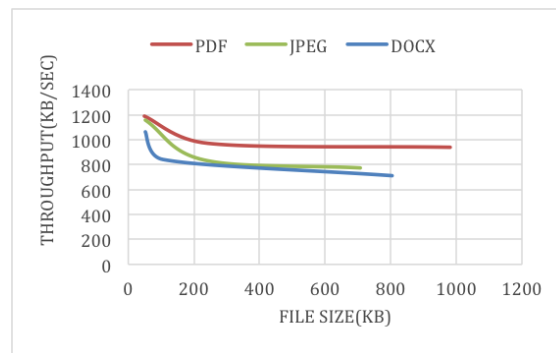


Figure 4.13: Average Throughput vs. File Size

Chapter 5: Conclusion and Future research

We presented a ESC3B protocol which addresses three main challenges in MLS cross-domain environments. We implemented three algorithms, EAFA, A2 and LC3 as part of ESC3B protocol and a user interface (UI). We tested and evaluated the correctness, stability and efficiency of these algorithms and integrated them to create the prototype. We successfully did the end-to-end testing of prototype using the UI.

In the future we can apply the ESC3B protocol with Hadoop to improve its security features. We can integrate our proposed ESC3B algorithms with the Hadoop to enable MLS cross-domain computing. We can also apply the hierarchical key structure in cloud computing to manage access to resources/services by users of various security levels.

Bibliography

- [1] C. Lamb and G. Heileman, “*Overlay Architectures Enabling Cloud Computing for Multi-Level Security Environments*”, 2012: IEEE 8th World Conference on Services.
- [2] Dan Boneh, “*Twenty Years of Attacks on the RSA Cryptosystem*”, 1999: Notices of the American Mathematical Society (AMS).
- [3] David Elliott Bell and Leonard J. LaPadula , “*Secure Computer Systems: Mathematical Foundations*”, 1973:MITRE.
- [4] Hash function, “ [http : //en.wikipedia.org/wiki/sha_256](http://en.wikipedia.org/wiki/sha_256)”
- [5] James Owen Sizemore, *Euler Phi Function*, 2012:University of Wisconsin, Madison, Lecture notes 9-11.
- [6] M. Kang and I. Moskowitz, “*A Data Pump for Communication*”, 1995: NRL Memo Report 5540-95-7771.
- [7] M. Kang and I. Moskowitz, “*A Pump for Rapid, Reliable, Secure Communication*”, 1993: Conf. Computer and Communication Security.
- [8] N. Swamy, M. Hicks, and S. Tsang, “*Verified Enforcement of Security Policies for Cross-Domain Information Flows*”, 2007:IEEE MILCOM.

- [9] OASIS, “*Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML)*”, 2005.
- [10] P. Steinmetz, “*User of Cross Domain Guards for CoNSIS Network Management*”, 2012: Communication and Information Systems Conference (MCC).
- [11] T. Hardjono, J. Seberry, *A multilevel encryption scheme for database security*, 1989: ACSC-12, University of Wollongong.
- [12] U. Dudley, *Elementary Number Theory*, 2008: Dover Books on Mathematics, Second Edition.

