cryptography

MDPI

Article

Trusted Time-Based Verification Model for Automatic Man-in-the-Middle Attack Detection in Cybersecurity

James Jin Kang [1,2,*], Kiran Fahd [1] and Sitalakshmi Venkatraman [1]

[1] Melbourne Polytechnic, Preston 3181, Australia; kiranfahd@melbournepolytechnic.edu.au (K.F.); sitavenkat@melbournepolytechnic.edu.au (S.V.)

[2] School of Information Technology, Deakin University, Burwood 3125, Australia

* Correspondence: jameskang@melbournepolytechnic.edu.au or j.kang@deakin.edu.au; Tel.: +61-439-192-855

Received: 9 November 2018; Accepted: 4 December 2018; Published: 5 December 2018

Abstract: Due to the prevalence and constantly increasing risk of cyber-attacks, new and evolving security mechanisms are required to protect information and networks and ensure the basic security principles of confidentiality, integrity, and availability—referred to as the CIA triad. While confidentiality and integrity can be achieved using Secure Sockets Layer (SSL)/Transport Layer Security (TLS) certificates, these depend on the correct authentication of servers, which could be compromised due to man-in-the-middle (MITM) attacks. Many existing solutions have practical limitations due to their operational complexity, deployment costs, as well as adversaries. We propose a novel scheme to detect MITM attacks with minimal intervention and workload to the network and systems. Our proposed model applies a novel inferencing scheme for detecting true anomalies in transmission time at a trusted time server (TTS) using time-based verification of sent and received messages. The key contribution of this paper is the ability to automatically detect MITM attacks with trusted verification of the transmission time using a learning-based inferencing algorithm. When used in conjunction with existing systems, such as intrusion detection systems (IDS), which require comprehensive configuration and network resource costs, it can provide a robust solution that addresses these practical limitations while saving costs by providing assurance.

Keywords: trusted time server (TTS); man-in-the-middle (MITM); Secure Socket Layer (SSL); Transport Layer Security (TLS); Secure Sockets Layer (SSL) time-based verification; inferencing schemes; cybersecurity; digital certificate; digital signature; inference algorithm

1. Introduction

A Digital Certificate (DC), also known as Secure Sockets Layer (SSL) certificate, has been used extensively in the cybersecurity domain and operates based on the public key infrastructure (PKI) with public key cryptography. It provides authentication, privacy, confidentiality, encryption, and digital signatures. It uses a private key for signing in and a public key for verification along with the identification (ID) of the certificate authority (CA) and the user who requested the DC. While it generally works well in providing authentication and ensuring the integrity of digital transactions, DCs are abused for the purpose of malware diffusion, cyber espionage, and sabotage of targeted or general famous sites. For example, state-sponsored hackers may have a great interest in demonstrating their capability by attacking a trusted CA, as highlighted by the case of the Comodo certificate hack. A well-known CA, Comodo, issued fraudulent DCs for nine websites, including Google, Microsoft, Skype, Yahoo, and other major sites [1]. An Iranian hacker, who was suspiciously sponsored by the state, hacked a registration authority (RA), a reseller of Comodo certificates, and claimed responsibility by posting the hacked incident at pastebin.com in 2011 [2]. This proves that even hardened security

measure and reputable DC providers can be vulnerable and are not free from man-in-the-middle (MITM) attacks.

While a digital signature (DS) provides the authenticity (A) and integrity (I) of the CIA triad, it does not provide confidentiality (C), as it uses a public key which is accessible by any user for decryption. PKI with symmetric key distribution is used to provide confidentiality. Multifactor authentication (MFA) has been introduced to harden security, with increased protection and prevention against any adversary attempting to gain access to a system. One-time passwords (OTP), timestamp, and nonce features are used to prevent replay attacks. Despite the aforementioned security measures and technologies, MITM attacks are known to be one of the most common and dangerous intrusions.

MITM attackers eavesdrop on the communication between two targets, and the attack is hard to detect. In particular, within an intranet environment, an adversary can freely surf or intercept confidential information. To prevent the attack, an intrusion detection/prevention system (IDS/IPS) can be used. However, several studies in the literature concur that such third-party solutions exhibit practical limitations due to persistent system complexities, escalating deployment costs, and several inconveniences, including business productivity losses due to high false alarms [3,4]. To address these issues with the aim of ensuring secure information communications, we propose a novel inference scheme using trusted time-based verification for automatically detecting MITM attacks.

The rest of the paper is organized as follows. Section 2 highlights selected related works and the gaps in the literature. Section 3 presents the research background describing the research objectives. In Section 4, we propose a novel approach to detect MITM attacks based on a transmission time-based verification approach. Section 5 describes the implementation of our model with the algorithms developed for this study. Finally, we provide the conclusions and future works in Section 6.

2. Related Works

Several studies using time-based verification for authentication have been reported in the literature. In the past, approaches to combine various pieces of information, such as a personal identification number and a verifier, were adopted to generate an authentication code within a single time interval [5]. To address their privacy drawbacks, more recently, the time-based one-time password (TOTP), as an extension of the HMAC-based OTP, was employed across a wide range of applications, such as remote virtual private network access, Wi-Fi logon, transaction-oriented web applications, and internet banking [6–8]. To enforce an MFA, a mobile App on a smart device is used to display a QR code which contains two time-based codes. These codes are then used for login verification, in addition to the existing login mechanism, by converting the OTP to app-based authentication [9,10]. These require manual intervention.

Network Time Protocol (NTP) has been used effectively to provide time synchronization between symmetric peers in a network. By using a basic pre-shared key scheme, authentication approaches have been explored, including the introduction of an autokey authentication protocol (RFC 5906) using PKI mechanisms [11]. An NTP Working Group released a draft for network time security for NTP using a mechanism of Transport Layer Security (TLS) and authenticated encryption with associated data (AEAD) for cryptographic security [12]. However, deploying such solutions are complex and expensive, leading to several practical limitations. Overall, existing approaches are quite inconvenient and lack automatic and seamless detection of MITM attacks. This forms the prime motivation of our research study.

While existing systems such as IDS/IPS may require complex configuration to the network by packet inspection and result in overhead, the proposed solution can be implemented with minimal impact to existing protocols. Existing approaches to the detection of such attacks have limitations, such as high implementation costs, unlike the proposed approach, which is advantageous as it does not require complex procedures and is inexpensive. The proposed inference procedure does not require modifications to existing protocols other than a minimal addition of a message prompting the subscription to the feature. When the proposed solution is used along with the existing IDS/IPS, it will

help security operators reduce false positives (FPs) because they are provided with assurance from the solution. An average security professional can only deal with a limited number of issues per day, whereas current IDSs in typical company networks can generate hundreds of alarms.

There have been developments on IP piracy prevention that can be used with hardware Trojan attacks. Yu Bi et al. tried to simulate camouflaging gates, polymorphic gates, and power regulators to prove the high efficiency of silicon nanowire field effect transistors (FET) and graphene symmetric tunneling (SymFET) in applications. They evaluated the unique properties of the new devices and their ability to protect circuit designs and counter IP piracy [13].

3. Research Background

In today's digital business world, secured and trusted information communication in a network is important, and this is offered by the SSL/TLS protocol security mechanisms. However, SSL/TLS relies on the correct authentication of the server using digital certificates, which could be tampered with by an adversary using several approaches that lead to MITM attacks. There are several real-life instances of illegitimately generated digital certificates. Furthermore, a hacker could intercept a conversation between the sender and receiver of a message to access the information for impersonation or tampering of contents. We studied this problem to arrive at a robust and easy-to-deploy solution, which is the primary focus of this paper.

We considered the abovementioned practical limitations in order to arrive at the overarching question, given below, that guided this research:

> Can we design an automatic trusted model that adopts an intelligent mechanism to employ time-based verification of sent and received messages to detect MITM attacks?

In order to answer the above research question, we describe a typical problem scenario with a proposed high-level solution approach given below:

> When a user requests the issuance of DC to a CA, the user may select the proposed option of using a timestamp check at a trusted time server (TTS), which will collect and verify the transmission time between when the message is sent and when it is received by the sender and receiver, respectively. The outcome of the verification is to advise the user of any suspicious MITM activity based on the difference between timestamps and preset thresholds, which can be maintained by ongoing updates. The automatic inference scheme is developed using a self-adaptive learning approach.

4. Proposed MITM Detection Model

We propose the use of time-based verification of transmission along with an inferencing database at a TTS to automatically and effectively detect suspicious activities from MITM attacks. We present our proposed model here by modeling the problem scenario and the types of MITM attacks detected by the model.

4.1. Modeling the Problem Scenario

Let us consider sender A (Alice) who transmits a message to receiver B (Bob) as shown in Figure 1. We assume that the information about the message, such as source and destination IP addresses, message ID, etc., can be simultaneously sent to a TTS which is trustworthy and records the sent timestamp, T_S. When the message arrives at the receiver, B sends the message information to the same TTS, which also records the received timestamp, T_R. The TTS then calculates the difference between Sent and Received time, which is denoted as T_D. This information is used to determine whether the message has been tampered with or not by comparing it with a predefined threshold value for tolerance along with appropriate learning algorithms within the Inference Database. This threshold table is adjusted continuously by the roundtrip time (RTT) obtained from the sender, who will ping

before and/or after the message transmission. If required, the receiver may send the ping results to the TTS to check the network status.

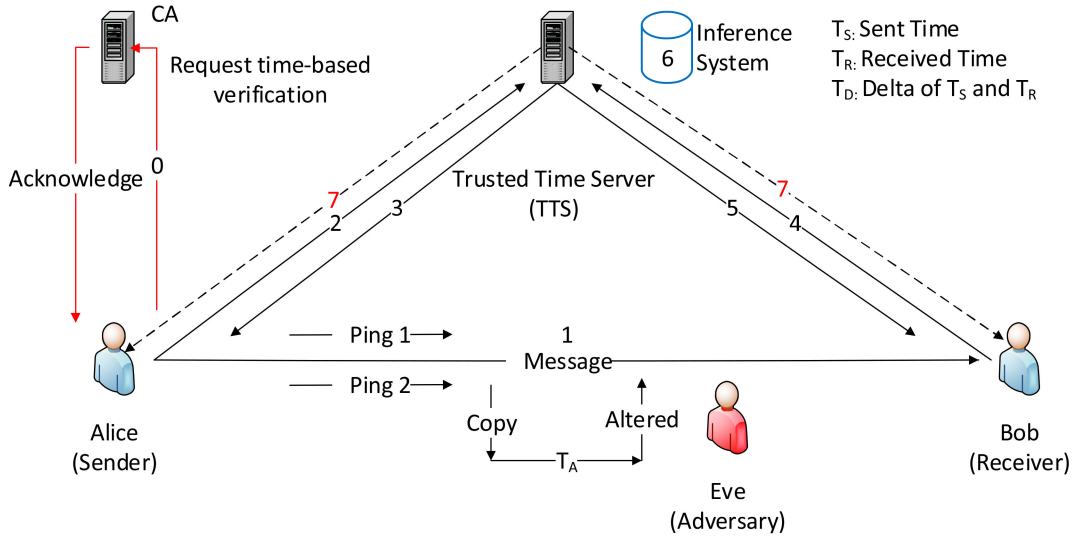

Figure 1. High-level overview of proposed man-in-the-middle (MITM) detection model.

When suspicious activity has been identified as an attack, the TTS adds the case to the existing database to record the intruder's details for future use. The TTS also creates and maintains the network status as separate data by the RTT within the network. All these data can be used to create a situation determination table, which will be used to determine a suspected intrusion.

We consider a possible scenario of MITM intrusion and describe a high-level overview of our proposed MITM detection model, as shown in Figure 1. The steps involved in our model are explained below:

Steps 1, 2, and 3: When A sends a message to B, the information about the message is also sent to a TTS. The server records the timestamp and responds to A with acknowledgment.

Steps 4 and 5: When the message is received by the recipient, B also sends the information about the message to the TTS. The server records the timestamp and responds to B with acknowledgment.

Steps 6 and 7: The TTS compares the sent and received timestamps against a preset threshold table to see whether the message has been delayed for longer than the expected transmit time period. If it detects a suspicious event by checking the database of situation determination, it sends an alert notification to the respective parties, A and B, about the possibility of an MITM attack. However, in order to avoid any false notification, the Inference Database is continuously updated with threshold table values using intelligent learning approaches. Sender A pings receiver B before and/or after the message transmission to see the response time, e.g., RTT, which will be used for updating the threshold table. If there was an MITM attack, there must be a sufficient time delay between the sent and received timestamp to arrive at a true anomaly, thereby confirming the inference.

4.2. Model Intelligence to Counter Evasive MITM Attacks

As shown in Figure 1, an MITM adversary (Eve) may try to emulate arbitrary latency so that additional time is earned to intervene delivery of the message. Once Eve inserts a synchronous pass through switch, it will slowly increase the latency time by merely storing and forwarding until both the sender and receiver are accustomed to the delay with the slightest degradation in latency, just as the adversary expected. To address this type of evasive MITM attack, our model exhibits intelligence using the inference engine to detect an abnormal delayed response time. TTS will infer the likelihood of the latency by continuously monitoring the network, as well as by learning through previous experiences and historical data.

An MITM adversary may adopt an evasive approach that involves manipulating timestamps. When the profile of messages from A and B have arrived at the TTS, our model is designed to intelligently check the validity of the timestamps, as these might have been replayed or invalid with elapsed time. To verify this, tools such as Casper [14] can be employed to validate the timestamps using the concept of age-stamps, where each timestamp actually specifies how long ago it was created.

Another evasive technique is one in which the adversary, such as Eve in Figure 1, monitors the communication channel and redirects all communication to fake sites to collect private information, such as the bank account details of Alice or Bob when they log into their bank's website. This can be avoided by using a TLS certificate, with additional checks on the use of the https application and further hardening with the proposed solution.

Our model makes use of intelligent learning approaches to counter the various evasive approaches that an MITM adversary may adopt. Furthermore, false inferences are avoided even with genuine network delays between A and B. For this, synchronization between the TTS and the network (including A and B) is important and can be achieved by employing network time security (NTS) in the network time protocol (NTP) [12]. Particularly when A and B are geographically far from each other and their communication must travel long distances, a false alarm may be triggered as a result of the delay from passing through various heterogeneous networks, such as cellular networks. This type of issues can be inferred intelligently by the TTS based on previous learning experiences gained in the network.

5. Implementation of Time-Based Verification Model

This section describes the model implementation and algorithms.

5.1. Model Implementation

When a DC issuance is requested, the user can select an option to use TTS-based timestamp verification using a new file type or extension. There are various file types and extensions that are used for DS between a server and a client, such as the personal information exchange format (.pfx)/public key cryptography standards (PKCS#12)/(p12) to export a certificate and its private key, a certificate signing request (.csr) to submit a request to CA, a Base64-encoded X.509 certificate (.cer or .crt) for a single certificate, a certificate revocation list (.crl) to identify relocated certificates, a certificate trust list (.stl), privacy-enhanced electronic mail (.pem) and private key (.key) etc. [15]. Our model implementation activates the feature of a new file type or extension, such as (.tbv) that can be used for time-based verification. For instance, when this type or extension is selected during a DC setup process between a server and a client, a TTS is appointed in the same way as the procedure for selecting a CA is selected, e.g., using (.stl). This way, all communications between the server and the client will send the message profile to the TTS to verify MITM intervention as an option.

To minimize the overhead of the network protocols, the TTS will only notify the related parties of suspicious activity when it has identified a likely attack. To improve the accuracy, the TTS is modeled to use an inferencing algorithm to determine a threshold customized for each network. Accordingly, the network will not be disrupted by an additional inquiry from the TTS unless it is necessary for further verification. When required, further clarifications, such as checking the timestamp validity of messages from A and B, may be required. Figure 2 depicts the message profile exchanged between the hosts and the TTS. The message profile details include the sent and received times transmitted by the sender and receiver hosts to the TTS. The TTS responds with timestamps and the inferred results to indicate whether the message is suspicious or not.

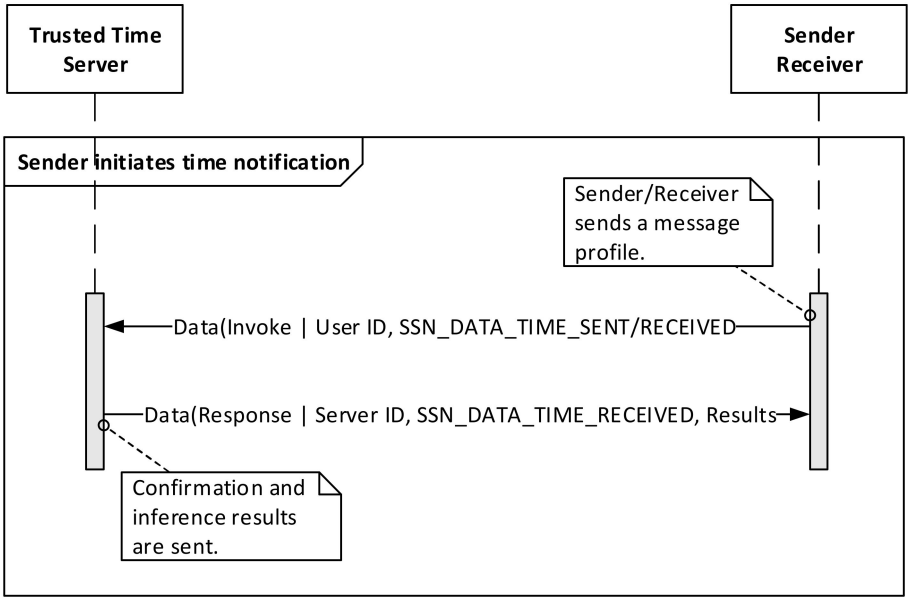

Figure 2. Message profile notification exchanged between sender/receiver hosts and the trusted time server (TTS).

5.2. Inference Engine Using Threshold Table

The inference engine of our model creates and makes use of a threshold table in the Inference Database to compare and assess a suspicious intrusion using learning-based inference rules. Several information details and parameters serve as inputs into the inference engine as follows:

- Ping results to and from A and B;
- Previous data and learning experiences;
- Network types (local, national, international, wired, wireless, cellular, metro, wide area network, etc.), which can be determined by source and destination IP that relates to geographic location;
- Current network status, e.g., normal, congested, abnormal etc.

Based on these inputs, a threshold value is decided with a level of margin, e.g., 1500 ms +/− 10%, as an outcome, which will be compared with the value of T_D (time difference between T_S and T_R). Table 1 shows an example threshold table created and referred to by the inference engine and TTS of our model to make an inference about an intrusion. The physical distance is calculated by the distance vector (DV) routing parameters, such as number of hops using the Bellman–Ford algorithm, and the logical distance is determined using well-established methods, such as the link state (LS) cost-based link state routing using the Dijkstra algorithm and hybrid Bellman–Ford–Dijkstra algorithm [16]. These algorithms aid in the reduction of convergence delays for link failure recovery [17]. The status of normal, congested, abnormal, and so on can be determined by a comparison with previous traffic data. Network diagnostic commands could be adopted, e.g., 'netstat –s', to display protocol statistics, including user datagram protocol (UDP), transmission control protocol (TCP), internet control message protocol (ICMP), and internet group management protocol (IGMP). The threshold margin is determined by averaged data learned previously. The margin indicates the possible latency learned from the previous experiences and historical data. The final threshold data can be averaged with P_d and L_d, along with other variables which are used to calculate the likelihood of an MITM intrusion by the inferencing algorithm. The network type is grouped based on the distance between the source and destination node. The outcome of the inferencing process will be an upper value of the standard deviation (SD) probability which has been predefined by the previous learnings from sample data. For example, 160 ms in Table 1 indicates the most optimal response time for network type B with distance vector-based protocols satisfying a 95% likelihood level. In other words, any delay longer

than this value can be regarded as a sign of unusual activity and potential intrusion, which would require further clarification by the inference engine to detect MITM attacks correctly.

Table 1. Threshold of tolerant response time.

Input Variables	Physical Distance: P_d (ms)—DV Based				Logical Distance: L_d (ms)—LS Based			
Network Type	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
Previous	100	200	350	500	50	300	400	600
Ping	50	150	400	700	20	200	360	550
Status	N	N	A	A	N	N	A	N
Margin (%)	5	10	15	17	5	7	9	15
SD Outcome	90	160	420	540	45	320	390	570

5.3. Time-Based Verification and MITM Detection Algorithm

In Figure 3, we depict the workflow for the implementation of our time-based verification model for MITM detection. The detailed workflow is given in Figure A1 in Appendix A.

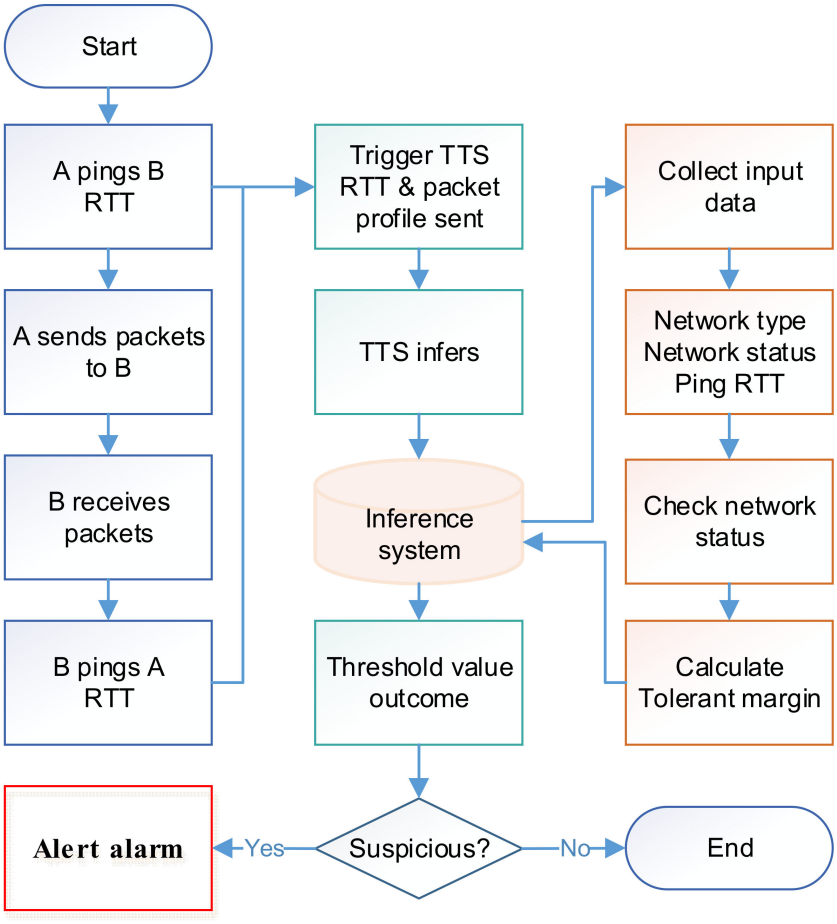

Figure 3. Workflow of time-based verification and MITM detection.

Algorithm 1 provides the proposed algorithm to detect MITM attacks by comparing timestamps of current activity, previous learnings, and historical data stored in the Inference Database. The detailed algorithm implementation using Java is given in Appendix B.

Algorithm 1. MITM detection algorithm.

```
Initialize: TP1 = 0, TP2 = 0, TD = 0
function createSentTimeStamp(MID,TP1)
      TS ← sent timestamp generated
   record MID, TP, TS in dbI database
end function
function updateReceivedTimeStamp(MID,TP2)
      TR ← received timestamp generated
   update the TR value based on MID in dbI database
end function
function calculatetimeDifference(TS,TR)
      TD ← TR − TS
   return TD
end function
function findThresholdValue(NT)
      find the threshold time for NT
   TH ←Avg(TH) of Physical distance—DV and Logical distance—LS
   TM ← Avg(TM) of Physical distance—DV and Logical distance—LS
   Ns ← 0.25(N),0.5(LC),0.75(MC),1(A)
   return TH, TM, NS
end function
function findStandarddeviation(NT)
      μ(mean) ← ((Tp) x (a%) + Tth x (b%) + Ns)/3
n←subtract the μ and square the result
   m← \frac{1}{3} \sum n
   σ = √m
   return μ, σ
end function
for each message send do
      A pings B
      TP1 ← ping response time
      TP1 = Tp1/2
      createSentTimeStamp(MID,TP1)
      B pings A
      TP2 ← ping response time
      TP2 = Tp2/2
      updateReceivedTimeStamp(MID,TP2)
        calculatetimeDifference(TS,TR)
        TH,TM ← findThresholdValue(NT)
        TP = (Tp1 + Tp2)/2
        update the threshold of tolerant response time table for Ping time
        findStandarddeviation
        σ = μ+ σ
        if ((TD > TP) AND (TD > (TP + TP * Tm/100)))
        if ((TD > μ) AND (TD > σ)) AND (TD > (σ + σ * Tm/100)))
          mark as a suspicious case
          Mwarn → sends a notification to A and B
        end if
      end if
   end for
```

The algorithm was developed in detail for the high-level solution of our proposed MITM detection model discussed earlier with reference to Figure 1. The algorithm compares values from the threshold table of tolerant response time in order to determine suspected intrusion effectively.

We provide the definitions of notations (in alphabetical order) in Table 2 used in the algorithm given in Algorithm 1:

Table 2. Definitions of denotation

db_I	Inference Database at the **trusted time server** (TTS)
N_S	Network Status—Current network status
N_T	Network Type—Current network type; network types include local, national, international, wired, wireless, cellular, metro, wide area network
M_{ID}	Message ID—Unique message identifier
M_{warn}	Warning Message sent to A and B
T_D	Time—Difference between T_S and T_R
T_{P1}	Ping Time—Response time when machine A pings machine B before sending the message to B
T_{P2}	Ping Time—Response time when machine B pings machine A after receiving the message from A
T_R	Received Time—Timestamp recorded by the TTS when machine B receives a message from machine A
T_S	Sent Time—Timestamp recorded by the TTS when machine A sends a message to machine B
T_{TH}	Threshold time—A threshold value that is decided based on the table of thresholds of tolerant response time, with the level of margin and network status based on the network type
T_M	Time margin—Level of margin which will be compared with the timestamp difference value T_D percentage; it is determined by averaged data learned previously.

Here, we describe the algorithm.

A pings B over a network of type N_T before the message transmission begins to see the response time (RTT). The algorithm calculates T_P1. Then, A sends a message to B, and all the required information about the message, including the message unique identifier M_{ID}, is sent to the TTS. N_T can be local, national, international, wired, wireless, cellular, metro, or wide area network. The TTS records the timestamp in Inference Database db_I when it receives the message from A, and it sends an acknowledgment confirmation to A. The algorithm generates a timestamp (T_S) when A sends a message to B. It then records M_{ID}, T_S, and T_P1 in the Inference Database db_I.

When B receives the message, it pings A, and T_P2 is calculated and the information about the message is also sent to the TTS. The algorithm generates the timestamp of message receipt T_R. It updates T_R and T_{P2} in the Inference Database db_I based on M_{ID}.

The algorithm then proceeds to calculate the time difference T_D. It also gets T_{TH}, which is a threshold value derived from the threshold table of tolerant response time, with a margin (T_M) level and network status (N_S) based on the N_T. T_{TH} and T_M are calculated from the average values of the physical distance (DV) and logical distance (LS) based on previous learnings. The mean (μ) and standard deviation (σ) are calculated continuously and updated. In addition, to provide intelligent inferences, the values of μ and σ are calculated with context-dependent weights associated with the calculated values for T_P, T_{TH}, and N_S. The a% weight of T_P and the b% weight of T_{TH} and N_S value would typically range between 0.25 and 1.

The algorithm finally performs various comparisons and checks for allowable threshold values of T_D, T_P, μ, and σ, with and without the level of margin T_M. If the values lead to the inference of an anomaly, indicating that the message delivery took much longer than the expected transmission time, the algorithm suspects an MITM attack and the TTS sends an alert notification to A and B. The Inference Database also gets updated with all the recorded values, which is useful for forming previous learnings.

5.4. Results and Analysis

We conducted a simulated experimental test using threshold values of tolerant response time from Table 1 and additional test data in order to verify and evaluate our proposed MITM detection. The detailed implementation of the algorithm is given in Appendix B. Table 3 provides the values of

the outcomes obtained for a network type A; the values are either calculated or retrieved from the threshold table in the database using our proposed inference engine.

Table 3. Experimental outcome for a network type A.

Network Type A	
networkTypevalue: → **A**	*Networkstatus:* Network status = N → *Networkstatus* = 0.25
Thresholdvalue: → (Physical distance value + Logical distance for **A** network type)/2 → 100 + 50 = 150 → 150/2 → *Thresholdvalue* = 75	*marginValue:* → (Physical distance value + Logical distance for **A** network type)/2 → 5 + 5 → 10/2 → *marginValue* = 5
PingRTT: → (Physical distance value + Logical distance for **A** network type)/2 → (60 + 20)/2 → *PingRTT* = 40	*mean:* → a% = 5%; b% = 7% → Arraylist = {(40) + 40 × 5%, (75) + 75 × 7%} → *mean* = **61.13**
standardDev: → a% = 5%; b% = 7% → ArrayList = {40 × 5%, 75 × 7%,0.25} → ArrayList = {2, 5.25,0.25} → *StandardDev* = **2.54**	*Mean + Standard deviation:* → 61.13 + 2.07 → **63.67**

Based on the predefined values of network type A, there are two cases to discuss, as follows:

(1) ***Nonsuspicious outcome*** of network type A, as shown in Table 4—the SD value was calculated and used to run the algorithm program as in Appendix B, and the results show that the variables are within the thresholds as previously defined.

Table 4. Stepwise verification of the algorithm for suspicious and nonsuspicious cases.

Case 1—Nonsuspicious Outcome	***Case 2—Suspicious Outcome***
***Timedifference* → 50**	***Timedifference* → 100**
→ **Compare** ((50 > 40) AND (50 > (40 + 40 * 5/100)))	→ **Compare** ((100 > 40) AND (100 > (40 + 40 * 5/100)))
→ **Compare** ((50 > 40) AND (50 > 42))	→ **Compare** ((100 > 40) AND (100 > 42))
→ return ***true***	→ return ***true***
→ *The Timedifference is greater than the PingRTT and the algorithm will perform the next Boolean comparison as below:*	→ *The Timedifference is greater than the PingRTT and the algorithm will perform the next Boolean comparison as below:*
→ **Compare** (50 > 61.13) AND (50 > 63.67)) AND (50 > (66.85))	→ **Compare** (100 > 61.13) AND (100 > 63.67)) AND (100 > (66.85)
→ return ***false***	→ return ***true***
Verification: As the above Boolean comparison returns a false, that means that the time difference is within the time limits based on the standard deviation of the average Ping RTT and Threshold time of Network type A. The time difference does not lead to the inference of an anomaly.	**Verification**: As the above Boolean comparison returns a true, that means that the time difference is outside the time limits based on the standard deviation of the average Ping RTT and Threshold time of Network type A. The time difference does lead to the inference of an anomaly.
Result: The TTS will mark this message as not suspicious and not send any notification to A and B.	**Result**: The TTS will mark this message as suspicious and send necessary notifications to A and B.

(2) ***Suspicious outcome*** of network type A, as shown in Table 4—the SD value is outside of the predefined threshold values, and the results show that the algorithm works properly, as it compares the data received with the predefined threshold data. This is shown in Appendix B.

5.5. Performance Measures

As a quality measure, the performance of our proposed algorithm requires continuous monitoring and automatic updates of the database. To achieve this, the following standard performance measures are used to attain the desired level of accuracy for detecting MITM attacks:

- True Positive (TP): Number of instances of correct inferences of an MITM attack possibility when an MITM attack is actually expected to happen;
- False Negative (FN): Number of instances of non-detection when an MITM attack has actually happened;
- False Positive (FP): Number of instances of wrong inferences of an MITM attack possibility when there is no MITM attack;
- True Negative (TN): Number of instances of correct inferences of no MITM attack possibility when there is no MITM attack happening;
- True Positive Rate (TP Rate): Percentage of correct inferences made. The TP rate is calculated as follows:

$$TP\ Rate = \frac{TP}{TP + FN}$$

- False Positive Rate (FP Rate): Percentage of wrong inferences made. The FP rate is calculated as follows:

$$FP\ Rate = \frac{FP}{FP + TN}$$

- Overall Accuracy: Percentage of correct inferences made. The overall accuracy is calculated as follows:

$$Overall\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

A True Positive (TP) is the most commonly used performance measure to evaluate the effectiveness of the detection model, as it is an indicator that the model can detect a suspected MITM and then raise an alarm before the attack happens. During real-time deployment, the higher the TP measure, the more accurate the model.

A False Negative (FN) is an indicator that the model does not detect the MITM attack when the attack exists. It is expected that in real-time deployments, very low FN values indicate a better performance of the model.

A False Positive (FP) is another commonly used performance measure to evaluate the level of tolerance and context-dependent inference exhibited by the model. It is an indicator of false detection when the network is conducting normal activities but appears to have an MITM attack due to unforeseen latency or network congestion. In real-time deployments, a low FP measure indicates a high level of learning and intelligence exhibited by the model.

A True Negative (TN) is an indicator that the model infers no suspected MITM activity when there is no MITM attack taking place. This measure also contributes to the overall accuracy and robustness of the model.

The True Positive Rate is an accuracy measure to determine the ability of the model to detect the MITM intrusions when MITM attacks exist. It is also referred to as the sensitivity. On the other hand, the False Positive Rate is used to evaluate the misjudgment ratio, which is also an important performance measure. It is also referred to as the specificity. In real-time deployments, a high TP Rate achieved along with a low FP rate indicates an effective model. Finally, the overall accuracy measure indicates the performance of the model in terms of correctness and robustness.

Another quality measure of the model based on a performance graphing method is called the receiver operating characteristic (ROC). It is a plot of the True Positive Rate on the Y-axis and False Positive Rate on X-axis. It can be used to evaluate different threshold schemes and compare the relative performance among different schemes to classify malware attacks. The overall accuracy measure is

based on one specific cutoff point and hence can vary for different cutoff points. On the other hand, ROC considers all cutoff points and plots the sensitivity (TP rate) and specificity (FP rate). The area under the ROC curve (AUC) depicts the relative trade-offs between true positives and false positives. Hence, the AUC was used to compare the performance of our model under different threshold schemes for malware classification. For instance, Figure 4 shows that the AUC under B is greater than that of A; thus, B exhibits a better performance than A.

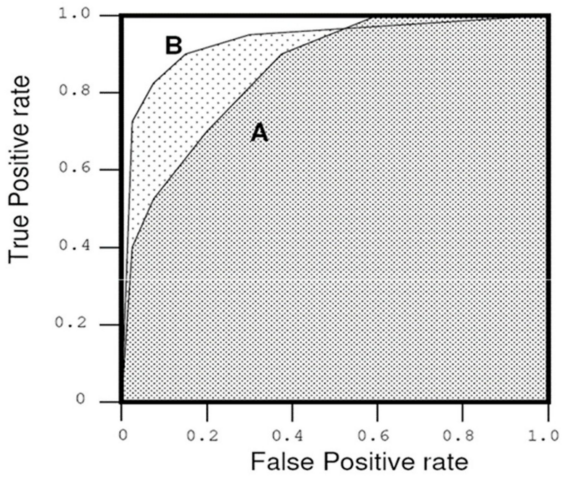

Figure 4. Comparison of performance measures using receiver operating characteristic (ROC) plots.

6. Conclusions and Future Works

Despite the application of secure SSL/TSL authentication techniques, MITM attacks are still prevalent. Security mechanisms such as SSL, TSL, HTTPs, and Certificate Authority are being enhanced for the provision of the CIA triad. However, evasive MITM attacks could even intercept certificates and forge them. In addition, existing solutions to the detection of MITM attacks have several practical limitations, including high implementation costs.

In this paper, we propose a trusted and effective model for detecting MITM attacks automatically using transmission time-based verification along with a novel learning-based inference scheme. The advantage of our model is that it requires neither complex system configurations nor expensive security implementations. Our inference algorithm works in the TTS without any need to modify existing protocols. By using a threshold table for tolerance in the Inference Database, a self-learning mechanism was established. Furthermore, the performance measures show our real-time deployment considerations for monitoring the effectiveness of the model.

The threshold table proposed as an example may not be accurate and could affect the accuracy of the detection model if applied straightaway. Future work will involve developing the threshold table for real-life implementation. Another interesting line of future work would be to locate an MITM adversary in the network. This could be possible by calculating distances from each node and the TTS and locating the intersection. For example, if an adversary is found with an unexpected location, such as a foreign country's IP address, the network operator may manually revise the policy to bypass the network from the point at which the adversary taps into the conversation.

When the TTS has built enough of a database knowledge base to compare with other servers in heterogeneous networks, it will be possible to develop and show pathways using data visualization techniques of the conversation paths. These paths can be classified with various credit levels, such as green, yellow, red, etc., so that network operators can use the information in their network topology configuration as additional attributes to the link costs or distances.

In order to measure the accuracy and performance and determine the usability and efficiency, a benchmarking with other solutions is required, such as IDS compared with the solution prototype.

Author Contributions: J.J.K. conceived of the idea, reviewed related works, designed the solution and application, wrote the original draft, and reviewed and edited. K.F. wrote the algorithms and flowcharts. S.V. wrote the performance measures, reviewed and supervised the overall quality of the article, and provided guidance of the contents.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interests.

Appendix A. Detailed Workflow for the Detection of MITM Attacks

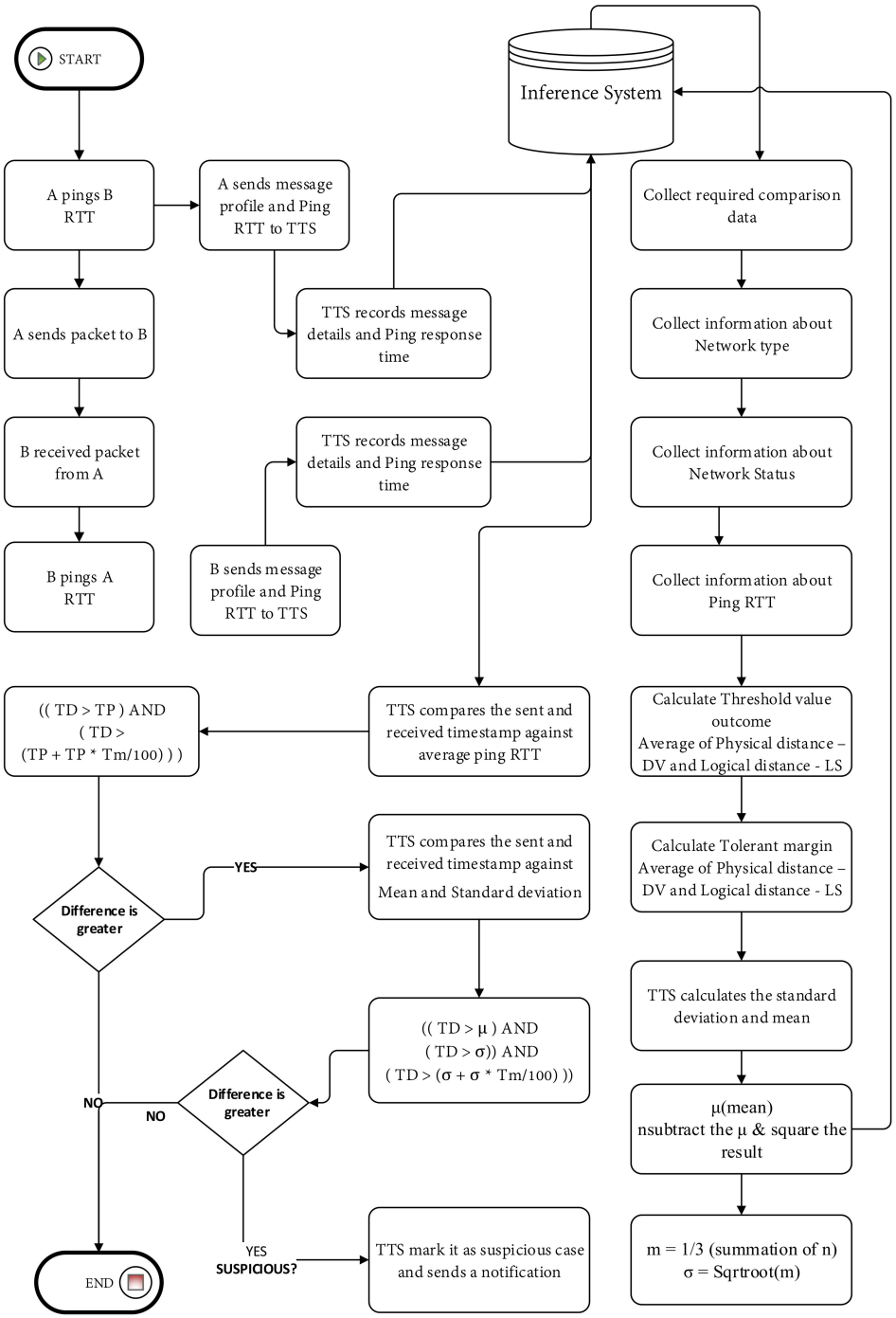

Figure A1. Workflow for the detection of MITM intrusion, which includes detailed scenarios and activity decision making.

Appendix B. Detailed Algorithm Implementation Using Java for Detection of MITM Attacks

Step 1 and 2: A pings B over the network of type NT before the message transmission begins so as to see the response time (RTT). The algorithm calculates TP1. Then, A sends the message to B, and all the required information about the message, including the message unique identifier MID, is sent to the TTS.

```
/**Calculates Response time when machine A pings machine B before sending the message to B */
String ipB= Machine B's IP Address;
InetAddress inetB = InetAddress.getByName(ipB);
double PingRTT, PingRTT1, PingRTT2 = 0;
double endTime = 0;
double beginTime = new GregorianCalendar().getTimeInMillis();
if (inetB.isReachable(5000)){
 endTime = new GregorianCalendar().getTimeInMillis();
/*PingTT1 stores Ping Time – Response time when machine A pings machine B before sending the message to B*/
 PingRTT1 = endTime - beginTime;
}

/** A sends message to B and to TTS */
String messageID = ;  //Message unique identifier (M_ID)
double PingRTT1 = ; //Ping response time calculate in the previous step
String ipTTS=  TTS's IP Address;
InetAddress inetTTS =
InetAddress.getByName(ipTTS);
int portTTS= TTS's Port Number;
private Socket socket;
this.socket = new Socket(inetTTS, portTTS);
this.socket.getOutputStream().write(messageID);
this.socket.getOutputStream().write(PingRTT1);
```

Step 3: The TTS records the timestamp in the Inference Database dbI when it receives the message from A and sends an acknowledgment confirmation to A. The algorithm generates a timestamp (T_S) when A sends a message to B. It then records M_{ID}, T_S, and T_{P1} in the Inference Database dbI.

```
/** JDBC connection steps to connect dbI: Inference Database at TTS and insert new record for M_ID.
  Assuming that the Inference database is MySQL database*/
String messageID = ; //Message unique identifier (M_ID) received from A in Step2
double PingRTT1 =  ; //Ping response time calculate in the step 1 and received by TTS in step 2
String dbDriver = "com.mysql.jdbc.Driver";
String dbUrl = "jdbc:mysql://localhost/DBI";
Class.forName(myDriver);
Connection connection = DriverManager.getConnection(myUrl, "username",
"password");
 Calendar calendar = Calendar.getInstance();
 java.sql.Timestamp timestampSent = new java.sql.Timestamp(calendar.getTime().getTime());
 /** Inserting a new record in a table named inferencetable in Inference Database for the current message with a unique identifier M_ID. The record values consist of M_ID, PingRTT1, Sent timestamp for message M_ID */
 String timestampsql = "INSERT INTO Inferencetable (mid,timestampsent,pingresponsetime1)
VALUES (?,?,?)";
```

```
PreparedStatement preparedStatement = connection.prepareStatement(timestampsql);
preparedStatement.setTimestamp(1, messageID);
preparedStatement.setTimestamp(2, timestampSent);
preparedStatement.setTimestamp(3, PingRTT1);
preparedStatement.executeUpdate();
```

Step 4 and 5: When B receives the message, it pings A; TP2 is calculated, and the information about the message is also sent to the TTS. The algorithm generates the timestamp of message receipt, TR. It updates TR and TP2 in the Inference Database dbI based on MID.

```
/**Calculates Response time when machine B pings machine A after receiving the message from A */
String ipA= Machine A's IP Address;
InetAddress inetA = InetAddress.getByName(ipA);
/*PingTT2 variable stores Ping Time – Response time when machine B pings machine A after receiving
the message from A*/
double PingRTT2 = 0;
double endTime = 0;
double beginTime = new GregorianCalendar().getTimeInMillis();
if (inetA.isReachable(5000)){
endTime = new GregorianCalendar().getTimeInMillis();
PingRTT2 = endTime - beginTime;
}

/** JDBC connection steps to connect dbI: Inference Database at TTS and updating the column values
for M_ID. Assuming that the Inference database is MySQL database*/
String messageID = ; //Message unique identifier (M_ID)
StringdbDriver = "com.mysql.jdbc.Driver";
String dbUrl = "jdbc:mysql://localhost/InferenceDB";
Class.forName(myDriver);
Connection connection = DriverManager.getConnection(myUrl, "username",
"password");
Calendar calendar = Calendar.getInstance();
java.sql.Timestamp timestampReceived = new
java.sql.Timestamp(calendar.getTime().getTime());
/** Updating an existing record in a table named inferencetable in Inference Database for the current
message with a unique identifier M_ID. The record will be update for the values of PingRTT2, timestamp for
receiving message M_ID */
String timestampsql = "UPDATE Inferencetable SET pingresponsetime2=?,
timestampReceived = ?  WHERE mid = ?  ";
PreparedStatement preparedStatement = connection.prepareStatement(timestampsql);
preparedStatement.setTimestamp(1, PingRTT2);
preparedStatement.setTimestamp(2, timestampReceived);
preparedStatement.setTimestamp(3, messageID);
preparedStatement.executeUpdate();
```

Step 6: The algorithm then proceeds to calculate the time difference TD. It also gets TTH, which is a threshold value derived from the threshold table of tolerant response time, with a margin (TM) level and network status (NS) based on the NT. TTH and TM are calculated from the average values of physical distance (DV) and logical distance (LS) based on previous learnings. The mean (μ) and standard deviation (σ) are calculated continuously and updated. In addition, to provide intelligent

inferences, the values of μ and σ are calculated with context-dependent weights associated with the calculated values for TP, TTH, and NS. The a% weight of TP and the b% weight of TTH and NS value would typically range between 0.25 and 1.

*/** JDBC connection steps to connect dbI: Inference Database at TTS and retrieving the saved information about Threshold time and time margin. Assuming that the Inference database is MySQL database*/*

*/** Threshold time – a threshold value that is decided from threshold of tolerant response time table with level of margin and network status based on the network type.*/*

```
double thresholdvalue = 0 ;
```

*/** time margin – Level of margin which will be compared with the timestamp difference value TD percentage and it is determined by averaged data learned previously*/*

```
double marginValue = 0 ;
```

/ Network Status – Current network status – Value will be 0.25(N),0.5(LC),0.75(MC),1(A) */*

```
double networkstatus=0;
  String networkTypevalue=""; // Local, National, International, Wired, Wireless, Cellular, Metro, WAN
  String dbDriver = "com.mysql.jdbc.Driver";
  String dbUrl = "jdbc:mysql://localhost/InferenceDB";
  Class.forName(myDriver);
  Connection connection = DriverManager.getConnection(myUrl, "username",
"password");
```

*/** Retrieving an existing record in a table named tolerantRTtable in Inference Database to get the value of Threshold time and Time margin based on value of Network Type – Current network type - Network types e.g. local, national, international, wired, wireless, cellular, metro, wide area network */*

```
   String thresholdvaluessql = "SELECT THPhysicaldistance, THLogicaldistance,
MarginPhysicaldistance MarginLogicaldistance ,networkstatus FROM tolerantRTtable WHERE
Networktype = ?  ";
   PreparedStatement preparedStatement = connection.prepareStatement(thresholdvaluessql);
  preparedStatement.setTimestamp(1, networktypevalue);
  Resultset rs = preparedStatement.executeQuery();
  if (rs.next()){
 PingRTT = rs.getInt("pingresponsetime");
```

*/** calculating an average value of Threshold time as an average of Physical distance and Logical distance of same Network type retrieved from database table based on the network type.*/*

```
thresholdvalue = (rs.getDouble("THPhysicaldistance") + rs.getDouble("THLogicaldistance"))/2;
```

*/** calculating an average value of Time margin as an average of Physical distance and Logical distance of same Network type retrieved from database table based on the network type.*/*

```
marginvalue = (rs.getDouble("MarginPhysicaldistance") +
rs.getDouble("MarginLogicaldistance"))/2;
```

*/** calculating the value of Network status based on value stored in database -0.25(N),0.5(LC),0.75(MC),1(A)*/*

```
  if (rs.getDouble("networkStatus").equals("N"))
  networkStatus = 0.25;
  else if (rs.getDouble("networkStatus").equals("LC"))
  networkStatus = 0.50;
  else if (rs.getDouble("networkStatus").equals("MC"))
  networkStatus = 0.75;
  else if (rs.getDouble("networkStatus").equals("N"))
  networkStatus = 1.00;
   }
```

```
/** Calculating Time difference of sent Timestamp and Received Timestamp and Average Ping RTT for Message unique identifier (M_ID)*/
 String messageID = ;  //Message unique identifier (M_ID)
double PingRTT = 0;  // Average Ping Time – Average of the Response time when machine A pings machine B before sending the message to B and when machine B pings machine A after receiving the message from A */
double timedifference=0; // Ping Time – Response time when machine A pings machine B before sending the message to B
 java.sql.Timestamp timestampSent,timestampRecieved;
String dbDriver = "com.mysql.jdbc.Driver";
String dbUrl = "jdbc:mysql://localhost/InferenceDB";
Class.forName(myDriver);
Connection connection = DriverManager.getConnection(myUrl, "username",
"password");
 Calendar calendar = Calendar.getInstance();
 java.sql.Timestamp timestampReceived = new java.sql.Timestamp(calendar.getTime().getTime());
/** Retrieving an existing record in a table named inferencetable in Inference Database to get the values of Ping RTT when machine A pings machine B before sending the message to B and when machine A pings machine B before sending the message to B, Sent timestamp, Received timestamp based on value of Message unique identifier (MID) */
String timestampsql = "SELECT pingresponsetime1, pingresponsetime2,
timestampReceived, timestampsent FROM inferencetable WHERE mid = ?  ";
 PreparedStatement preparedStatement = connection.prepareStatement(timestampsql);
 preparedStatement.setTimestamp(1, messageID);
 Resultset rs = preparedStatement.executeQuery();
 if (rs.next()){
/** Calculating Average Ping Time Response for Message unique identifier (M_ID)*/
PingRTT = (rs.getDouble("pingresponsetime2")+ rs.getDouble("pingresponsetime1"))/2;
/** Calculating Time difference of sent Timestamp and Received Timestamp for Message unique identifier (M_ID)*/
   timedifference = rs.getDouble("timestampReceived")- rs.getDouble("timestampSent")
;
 }

/** Calculating Mean and Standard Deviation of Ping RTT , Threshold value and Network status for a particular Network type*/
/** Threshold time – a threshold value retrieved earlier based on the network type.*/
double thresholdvalue = 0 ;
/** time margin – Level of margin calculated retrieved previously based on Network type*/
double marginValue = 0 ;
/* Network Status – Current network status – Value calculated earlier */
double networkstatus=0;
double sum,mean,mSD,standardDev=0;  //variables to store Mean and Standard deviation */
double PingRTT;  // Average Ping Time Response for Message unique identifier (M_ID) calculated in previous step*/
String networkTypevalue="";  // Local, National, International, Wired, Wireless, Cellular, Metro, WAN
double apercent,bpercent;
/*Calculating Mean*/
 mean= ((PingRTT + PingRTT*apercent) + (thresholdvalue +thresholdvalue*bpercent))
/2;
```

```
// Creating an Array to calculate find Mean and Standard deviation.

  double valueArray = { PingRTT*apercent , thresholdvalue*bpercent , networkstatus
};
  int length = valueArray.length;
  for (double num:  valueArray){
    sum+=num; }
  mSD=sum/length;
for (double num:  valueArray){
    standardDev +=Math.pow(num - mean,2);
  }
   /*Calculating Standard deviation-*/
  standardDev = Math.sqrt(standardDev/length);
```

Step 7: The algorithm finally performs various comparisons and checks for allowable threshold values of TD, TP, μ, and σ, with and without the level of margin TM. If the values lead to the inference of an anomaly, indicating the message delivery took much longer than the expected transmission time, the algorithm suspects an MITM attack and the TTS sends an alert notification to A and B.

```
/**Comparison to find out the suspicious message due to MITM attack*/
String Warningmessage= "Suspicious message received"; // Warning Message sent to A and B
from TTS
/**Comparing time difference of time of sent and time of receiving message is greater than the Average or time
margin valued time difference percentage* /
if (( timedifference > PingRTT ) && ( timedifference > (PingRTT + PingRTT *
marginValue/100) ) ) {
    /**Comparing time difference of time of sent and time of receiving message is greater than
    the Mean value or greater than the Standard deviation or time margin valued Standard
      deviation percentage* /
    if ((timedifference > mean ) && ( timedifference > standardDev)) AND
     (timedifference > (standardDev + standardDev* marginValue/100))){
    /**If comparison indicates that the message delivery took much longer than the expected
      transmission time, the algorithm suspects an MITM attack and the TTS sends an alert
      notification to A and B* /
      String ipA= A's IP Address;
      String ipB= B's IP Address;
      InetAddress inetA = InetAddress.getByName(ipA);
  int portA= A's Port Number;
  InetAddress inetB = InetAddress.getByName(ipB);
  int portB= B's Port Number;
   private Socket socket1, socket2;
   this.socket1 = new Socket(inetA, portA);
   this.socket2 = new Socket(inetB, portB);
   this.socket1.getOutputStream().write(Warningmessage);
   this.socket2.getOutputStream().write(Warningmessage);
}
}

  String ipB= Machine B's IP Address;
  InetAddress inetB = InetAddress.getByName(ipB);
```

```
double PingRTT, PingRTT1, PingRTT2 = 0;
double endTime = 0;
double beginTime = new GregorianCalendar().getTimeInMillis();
if (inetB.isReachable(5000)){
  endTime = new GregorianCalendar().getTimeInMillis();
  PingRTT1 = endTime - beginTime;
}

    String messageID =  Message unique identifier (MID)
double PingRTT1 =  Ping response time calculate in the previous step;
String ipTTS=  TTS's IP Address;
InetAddress inetTTS =
InetAddress.getByName(ipTTS);
int portTTS= TTS's Port Number;
 private Socket socket;
 this.socket = new Socket(inetTTS, portTTS);
 this.socket.getOutputStream().write(messageID);
 this.socket.getOutputStream().write(PingRTT1);
```

References

1. Bright, P. Independent Iranian hacker Claims Responsibility for Comodo Hack. Available online: http://www.wired.com/threatlevel/2011/03/comodo_hack (accessed on 11 October 2018).
2. Comodo Hacker, A Message from Comodo Hacker. Available online: https://pastebin.com/74KXCaEZ. (accessed on 11 October 2018).
3. Scarfone, K.; Mell, P. Guide to Intrusion Detection and Prevention Systems (IDPS). *NIST Spec. Publ.* **2007**, *800*, 94.
4. Agarwal, N.; Hussain, S.Z. A Closer Look at Intrusion Detection System for Web Applications. *arXiv*, arXiv:1803.06153.
5. Brainard, J.G.; Kaliski, B.S., Jr.; Rivest, R.L. Method and apparatus for performing enhanced time-based authentication. U.S. Patent No. US7363494B2, 22 April 2008.
6. M'Raihi, D.; Machani, S.; Pei, M.; Rydell, J. Totp: Time-Based One-Time Password Algorithm. Available online: https://www.rfc-editor.org/rfc/rfc6238.txt (accessed on 11 October 2018).
7. Uymatiao, M.L.T.; Yu, W.E.S. Time-based OTP authentication via secure tunnel (TOAST): A mobile TOTP scheme using TLS seed exchange and encrypted offline keystore. In Proceedings of the 4th IEEE International Conference on Information Science and Technology (ICIST), Shenzhen, China, 26–28 April 2014; pp. 225–229.
8. Hsieh, W.-B.; Leu, J.-S. Design of a time and location based One-Time Password authentication scheme. In Proceedings of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC), Istanbul, Turkey, 2011; pp. 201–206.
9. Kemshall, A.C. Time-based authentication. U.S. Patent US9363077B2, 7 June 2016.
10. Oberheide, J.; Goodman, A.; Czub, C.; Garrity, P. System and method for converting one-time passcodes to app-based authentication. U.S. Patent US20160197914A1, 22 May 2018.
11. O'Donoghue, K. A New Security Mechanism for the Network Time Protocol. *IETF J. Internet Secur.* Available online: https://www.ietfjournal.org/a-new-security-mechanism-for-the-network-time-protocol/ (accessed on 31 October 2017).
12. NTP Working Group. Network Time Security for the Network Time Protocol. Internet Engineering Task Force, 2017. Available online: https://tools.ietf.org/id/draft-ietf-ntp-using-nts-for-ntp-10.html (accessed on 11 October 2018).
13. Bi, Y.; Gaillardon, P.; Hu, X.S.; Niemier, M.; Yuan, J.; Jin, Y. Leveraging Emerging Technology for Hardware Security—Case Study on Silicon Nanowire FETs and Graphene SymFETs. In Proceedings of the 2014 IEEE 23rd Asian Test Symposium, Hangzhou, China, 16–19 November 2014; pp. 342–347. [CrossRef]
14. Lowe, G. Casper: A compiler for the analysis of security protocols. *J. Comput. Secur.* **1998**, *6*, 53–84. [CrossRef]

15. Panday, K.K. Various SSL/TLS Certificate File Types/Extensions. Available online: https://blogs.msdn.microsoft.com/kaushal/2010/11/04/various-ssltls-certificate-file-typesextensions/ (accessed on 11 October 2018).
16. Dinitz, Y.; Itzhak, R. Hybrid Bellman–Ford–Dijkstra algorithm. *J. Discret. Algorithms* **2017**, *42*, 35–44. [CrossRef]
17. Waleed, S.; Faizan, M.; Iqbal, M.; Anis, M.I. Demonstration of single link failure recovery using Bellman Ford and Dijikstra algorithm in SDN. In Proceedings of the 2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT), Karachi, Pakistan, 5–7 April 2017; pp. 1–4.