Soft Computing (2021) 25:12343–12356
https://doi.org/10.1007/s00500-021-05928-6

FOCUS

Securing data in transit using data-in-transit defender architecture for cloud communication

Keerthana Nandakumar[1] · Viji Vinod[2] · Syed Musthafa Akbar Batcha[3] · Dilip Kumar Sharma[4] · Mohanraj Elangovan[5] · Anjana Poonia[6] · Suresh Mudlappa Basavaraju[7] · Sanwta Ram Dogiwal[8] · Pankaj Dadheech[9] · Sudhakar Sengan[10]

Accepted: 31 May 2021 / Published online: 10 June 2021

Abstract
The advent of cloud infrastructure in which third-party cloud services may retain sensitive consumer and company data in storage environments underlines the need to advocate for encryption and multi-tenant shared processing as a primary security mechanism. Digital information movement, storage, and processing are widely defined in terms of "Data in Motion," "Data at Rest," and "Data in Usage," respectively. The implementation of security methods for each of these states can be viewed similarly. Transit data applies to data when being moved from one source of data to another. Transit data contains data sent across the network from back-end clients, programs, and repositories. There could be two data centers inside the same organizational network in the cloud, as a member of completely separate networks. This paper presents a novel architecture data-in-transit defender (DiTD), to protect data in transit; DiTD provides a novel security framework based on high-performance cloud computing. This protocol enables more efficient use of the key strength and time of symmetric block encrypted data, public-key cryptography (PUKC), cryptographic hash, and brief key exchange function.

Keywords Data in transit · Cloud security · Encryption · Ciphertext · Cloud service

1 Introduction

Over the past decades, the world's Web has undergone an unprecedented increase in hackers, malware, ransom wares, and other harmful bugs or groups who are actively attempting to find a way to access user data. This state goes without saying that security has become one of the most critical activities that can be addressed, irrespective of the position we usually perform (Adrian et al. 2015). The general necessity to avoid unauthorized access to confidential, private/otherwise vital details is something every end-users, database operators, system managers, and so on can acknowledge: The disagreements are primarily linked to what we ought to secure and how we can do it.

The process of deciding the best route to secure our data always includes a well-conducted risk analysis supported by a cost–benefit analysis, which is an effective model for the order to assist us in identifying the appropriate technological and operational measures to be followed in our given scenario (Aviram et al. 2016). The control system or processor must use adequate processes and technologies to maintain a minimum level of security to minimize the costs, taking into account communications systems, the cost of implementation and design, the scope, description, and objective, the threat of varying probability and the impact on human freedoms and privileges.

As the name suggests, in-transit data can be treated just like a medium of transmission: A perfect illustration of in-transit data is a standard Web page that we receive from the internet while we browse the internet. In a nutshell, this is what happens under the hood (Flavel et al. 2015):

1. We submit a request for HTTP (or HTTPS) to the server, which runs the Web site we use.
2. The Web server acknowledges our request, processes it by identifying the (static/dynamic) information that we have requested, and then sends it to us as an HTTP (or HTTPS) address over a specific TCP port (usually 80 for HTTP and 443 for HTTPS).

Communicated by Vicente Garcia Diaz.

Extended author information available on the last page of the article

3. Our application, typically a Web browser such as Google Chrome and Firefox receives the response(s) in HTTP, stores it in their internal cache, and displays it to us.

Now, let us assume for granted that both the server and the client at rest introduced a high degree of data encryption: This ensures that the first and fifth states are internally secure, as any attempt at penetration against encrypted data is created. However, the third condition is that the data-in-transit may or may not be authenticated based on the application protocol. The client currently utilizes the data to be transmitted.

This is what usually happens under the hood when the HTTP protocol is being used:

As we can see, the security risks become quite clear. When the Web server reads the incoming request and decrypts the data, the submitted data become transparent, and the route used to send it to the Web client (HTTP) is not encrypted (Liu et al. 2015). As an outcome, any unauthorized party that effectively performs an attack (Fig. 1) directly accesses our encrypted information.

2 Related works

Additional conventional cryptographic protocols (e.g., TLS/SSL) do not always address the increasing security requirements in cloud communications. These motivations are mainly related to maintaining compatibility with the middlebox, backward compatibility with older systems, reduction due to non-availability of the selected protocol version, and some recent attacks (e.g., BEAST, CRIME, DROWN, WeakDH, and SSLv3 fallback, BREACH, POODLE, Logjam, and ROBOT threats) (Cangialosi et al. 2016).

Recently, the final draft of the TLS *v1.3* was published. This reports some enhancements in terms of security and efficiency over TLS *v1.2*. TLS *v1.3* excludes both existing symmetric encryption processes and the static RSA and the Diffie–Hellman cipher suites from helping. This is applied to the base requirements DHE (Bittau et al. 2010). It also uses only related data Authentic Encryption with the Accompanying Data (AEAD) processes to authenticate the encryption.

However, it has specific weaknesses. In TLS *v1.3*, the first two handshake round-trip messages are combined into a single round-trip address. The mixed message contains the approved cipher suites details, the network key exchange details, and the *"Client_Hello_Msg"* response together in the unencrypted form (Cramer and Shoup 2004; Kodali and Sarma 2013). Most notably, both transactions are conducted in unencrypted form until the Web gets a *"Server_Hello_Msg"* reply. The Client key exchange knowledge is one-half of the key exchange process provided by the server-side algorithm random presumption. As a result, if the server does not accept the algorithm or does not send any key share data, the user creates and re-submits key exchange data using the recognized algorithm, helping to reduce the round-trip time. Along with the above improvements, it also uses Pre-Shared Key (PSK) cipher suites. The Redundant messages like *"Change_Cipher_Specification"* are often omitted, and thus, it keeps backdoor support available for the middlebox applications (Tillich and Großschädl 2005).

TLS (Transport Layer Security), as well as related protocols such as QUIC, depending on the third-party repositories—Certificate Authorities (CA), is used to

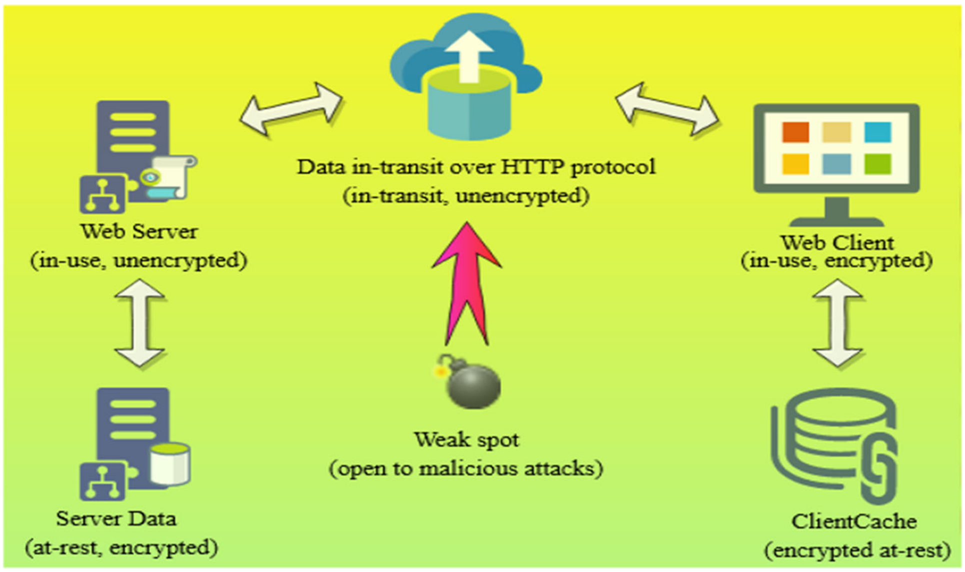

Fig. 1 Weak spot (open to malicious attacks)

guarantee the authentication of the cryptographic keys shared between two exchanging parties (e.g., client and server), thereby defending beside successful Man-In-The-Middle (MITM) assaults. However, we have seen several issues in this program in recent years (Megouache et al. 2020): (a) CA's need to be tricked into granting certificates to the untrusted user; (b) CA servers have compromised; (c) the process to revoke weak or stolen certificates is troublesome; and (d) CA transfers the hidden keys to another possibly dangerous user. Consequently, the security offered by current TLS, contrary to successful MITM attacks, is maximally debatable.

On the other side, opportunistic encryption protocols like Tcpcryp offer an enhanced security standard for Internet communications. Instead of utilizing any cryptography, these protocols encrypt and authenticate the data without allowing communicators to authenticate each other's public keys (PBK). It offers security against proactive threats but leaves the door open for aggressive others (Failed 2020). Essentially, this is equivalent to utilizing a corrupted CA certificate or the self-signed one.

3 Proposed methodology

Throughout this segment, we suggest extensive stable data-in-transit defender (DiTD) architecture for cloud communication. This design effectively reduces cloud communications' risks that occur within cloud entities. DiTD guarantees security Cloud Service Providers (CSPs) (Mohiuddin and Almogren 2020) and Cloud Users (CUs) in-transit data and validity. It has no middlebox or backward compatibility. If the two parties interact using the NIST-recommended cipher suites, then the protected connection cannot be created. They carry out Man-In-The-Middle (MITM) vulnerability monitoring (including eavesdropping, name spoofing, sniffing, code tampering), confidential knowledge leakage, compromised-key, repeat, repudiation, and device hijacking assaults (Elazhary 2019). But we are demonstrating that this design prevents such attacks effectively. DiTD defends the cloud contact networks with the significantly less overhead agreement and capacity, fair resource utilization, and better access than standard authentication protocols (e.g., TLS *v1.3*) (Almogren 2019).

Our key contribution in this work is a comprehensive stable infrastructure for cloud communication named DiTD. More precisely, the article refers to the following:

- DiTD offers a novel security framework based on the high-performance cloud. The power and speed of symmetric block encryption, PUKC, cryptographic hash, and brief key exchange function are used efficiently by this protocol.
- Using modern and concise message architectures facilitates reconnection, stable session configuration, and data transfer. Such message architectures support achieving reduced bandwidth utilization and reasonable memory use associated with the TLS *v1.3* (the current regular variant of SSL successors) and implement other communication protocols within it (Shailendra et al. 2018).
- DiTD guarantees data-in-transit confidentiality and all related hidden keys. By conducting Secured Key Exchange (SECKEX) for each client and encrypting the client with a fresh hidden key, it retains Perfect Backward Secrecy (PBS) and Perfect Forward Secrecy (PFS)
- Secured Key Exchange (SECKEX) is a key exchange protocol, expanding key exchange Diffie–Hellman. By exploiting cloud communication channels between interacting parties, SECKEX offers opportunistic encryption with partial security, contrary to successful MITM attacks. Therefore, our protocol has all the usability advantages of opportunistic encryption while providing security against many active attackers.

4 DiTD architecture

This segment discusses, in detail, the suggested secure infrastructure for cloud communication. Within the following segment, we address the nature of this architecture and its various stages of communication. In this, we clarify the chain of activities executed at all ends of users and servers.

4.1 Design specification

DiTD tackles data-in-transit security in cloud computing. Utilizing a modern Key Management Server (KMS) framework ensures the validity of cloud entities (Gawannavar et al. 2015; Veerabathiran et al. 2020). The KMS is built to protect, delete, and securely spread public root keys. DiTD successfully incorporates symmetric block encryption intensity and time, PUKC, cryptographic hash, and ephemeral key exchange system. Symmetric encryption helps to guarantee confidentiality, cryptographic hashing helps to ensure integrity, and PBK cryptography ensures trustworthiness and non-repudiation. It includes these four critical security mechanisms into communication systems. A Cloud Service User (CSU) may provide a safe channel of contact with Cloud Front End (CFE), and it ensures that data and cryptographic keys are still secure.

No long-term keys are used on the network. The session is encrypted along with a new secret key guaranteeing Perfect Forward Secrecy (PFS). DiTD is specific to interactions focused on both TCP and UDP. It operates in the layer of operation. It can therefore be effectively implemented in most of the protocols and application networks. DiTD utilizes seven different message architectures, which are extremely compact: (i) To publish (PUB), (ii) To acknowledge (ACK), (iii) To reconnect (RECON), (iv) For request (REQ), (v) For response (RES), (vi) To expire (EXP), and (vii) To find the error (ERR). We render DiTD more effective bandwidth usage, memory usage, and integration with existing protocols. Thus, these message architectures enable safe session configuration, reconnection, data transfer, and cloud-based error handling (Hamad et al. 2020).

The design consists of six separate communication phases: registration, configuration, creation of a session, transmission of data, termination, and reconnection. Throughout the registration process, the cloud organizations must first report the root PBKs to the Key Management Server (KMS). In the setup process, the partial cryptographic key pairs and the Hash Function (HF) are designed to create an encrypted session before any cloud user chooses to communicate to the private cloud. Instead, all organizations share their shared temporary keys signed by their respective private root key. Hybrid-crypto protocol secures the key exchange of temporary PBKs using AES/ECC Code for device encryption and SSDH/ECDH for key exchange of the session creation process.

After that, all organizations use SECKEX to produce a standard symmetric encryption key. Then, in the data transmission process, they continue sending coded signed data to each other. The cloud server dismisses the link, called the termination process, after successfully transmitting the answer payload to the cloud customer. At this stage, the server retains the session details protected before the client expires. The cloud user submits a reconnection request during that time and recovers the encrypted session for further data transfer, called the reconnection process. Figure 2 Illustrates specific interaction steps used by DiTD that are addressed in depth in the following section.

4.2 Registration phase

Before every communication, all cloud entities are registered with root PBKs on the Key Management Server (KMS) (Jurcut et al. 2020) of the DiTD. PBKs for the KMS will consistently be implemented in the cloud agency applications to guarantee the data's security and reliability between the KMS and the server. At this stage, it is assumed that the KMS, along with all its correspondences (Key Registration, Key Revocation, and Key Distribution), is secured.

4.2.1 DiTD's key management server

A Data Encryption Key (DEK) is the device used to encrypt the data in a piece. Such keys are placed close to the data they encrypt, leading to the need for low latency and high availability. The DEKs are authenticated with a Standard Encryption Key (SEK) (or "wrapped by"). For Cloud Platform providers, one or more KEKs exist (Vijaya Kumar et al. 2020). Such KEKs are centrally maintained in Key Management Server (KMS), a server specifically designed to hold keys. With a reduced number of KEKs than DEKs and a single key management tool, DiTD's encryption data exchange is more scalable. It enables user access from a central point to be managed and regulated.

4.3 Setup phase

In the case of Cloud service Instance (CI), CI is started at the very beginning. Nevertheless, for the Cloud User (CU), interactions with the Cloud Front End (CFE) server begin when a new cloud link is established. A growing cloud entity creates a couple of temporary PUK-PK pairs during this process. One key pair (AES/ECC) is the reliability and validity of the payloads. For protected key exchange, the other leading pair is (SSDH/ECDH). Every cloud entity often initializes the cryptographic HF as defined in the specification.

4.4 Session creation phase

When CU first links with CI, a temporarily authenticated connection between CU and CI is initialized. A pair of messages are exchanged among them during this process. In the short session, secured by a 64-byte hashed session key, all parties store the pair of PBKs from the other side. Then, they produce a specific secret key for the data transmission process to begin. After each valid transaction, the 64-byte hashed session key (request-response) is modified. The CU regularly gets a modified session key concealed within the encrypted response. Once the session ends, all the established PBKs are immediately deleted, and a new secret key is created.

4.5 Data transmission phase

Once the protected session has been created, all parties use the proposed 2-TCA algorithm to execute hybrid block encryption to preserve the payloads' secrecy for request and answer. The agreed temporary key pair (AES/ECC) is used during the session to conduct payload signing and

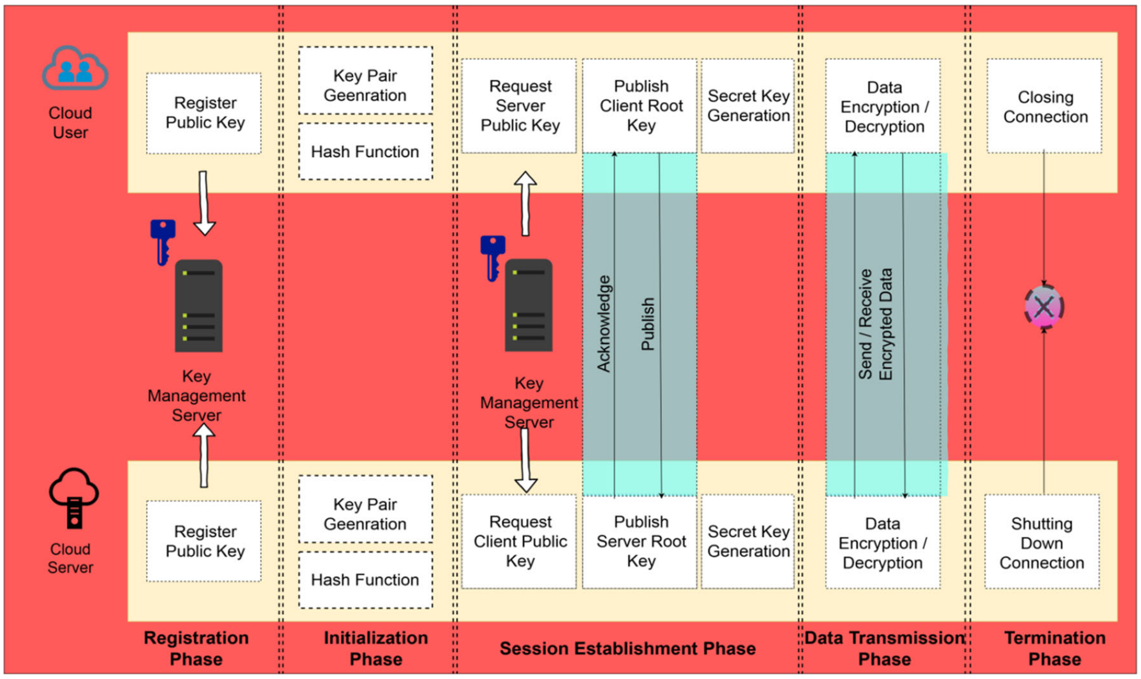

Fig. 2 DiTD architecture communication phases

verification, which guarantees the payload's validity and legitimacy. Each signing procedure carried out in this architecture requires timestamps to defend in contrary to replay attacks. The cloud-focused cryptographic HF is mainly used to secure the privacy of the data during this process.

4.6 Termination phase

During this step, the contact channel is terminated when the CI successfully sends the encrypted response back to CU. The current session stays valid until it expires for reconnection.

4.7 Reconnection phase

This process is not seen directly in Fig. 2. Within the design, it has an implicit operation. When the CU connects the server again and directs a legitimate reconnection (RECON) packet with the last obtained session key during the termination process, the encrypted link is reset among CI and CU. CFE provides a database of session key CIs. It reconnects CU to correct CI, depending on the session key. All sides use the PBK pair previously exchanged, and the specific hidden key held. So, it is not appropriate to re-key the block cipher throughout the session.

5 The proposed 2-tier cryptography algorithm (*2-TCA*)

Draft 2-TCA is discussed in this segment. This 2-TCA incorporates a modern process of integrating symmetric and asymmetric methods across two parallel processes. Such iterations mitigate the drawbacks of current hybrid algorithms by attaining a high degree of reliability without increasing execution time.

5.1 Encryption process

The Data in the Encryption is broken into n blocks, Bi. Every block is composed of 128 *bits*. This is separated into two sections of Msg_i $(0{:}{}^{n}/_{2}{-}1)$ and Msg_i $({}^{n}/_{2}{:}n - 1)$. If n is not an integer number and has a fraction, then the 2-TCA algorithm uses padding with null to be 128 bits for the last row. The encryption cycle is divided into two phases: The first ${}^{n}/_{2}$ fragments are secured using the hybrid encryption method (ECC and AES) in Phase I. The ECC algorithm is used to protect the secret key; subsequently, it is the PBK's first stable algorithm.

ECC requires smaller key sizes than the other methods, leading to fewer memory spaces, depending on the mathematical question. The ECC solved entirely exponential rather than the sub-exponential for the other public-key schemes (Hamad et al. 2020). ECC allows contact nodes to accommodate the more considerable number of queries for the lowest number of packets lost.

Meanwhile, ECC uses more energy than the symmetrical process; the AES system's usage decreases resource consumption and improves the device's efficiency. Applying AES with ECC needs to save up to 25% more energy and optimize the encryption and decryption procedures by approximately 20%. The first $^{n}/_{2}$ number of blocks is encrypted as subsequent: 'm_i is encrypted using AES through the key K_i, an undisclosed key of AES encryption procedure with 128 bits. K_i is encrypted through ECC for producing K_j with the length L.

$$m_i = \sum_{i=0}^{i=\frac{n}{2-1}} (Bi), 0 \le i \le \frac{n}{2-1} \tag{1}$$

$$K_j = \mathrm{ECC}_{\mathrm{enc}}(\mathrm{TC}_{\mathrm{PK}}, k_{i-1}), 0 \le j \le L-1\ldots. \tag{2}$$

The ECC utility is $\mathrm{ECC}_{\mathrm{enc}}$; the input ciphers with the Trust Center Public Key (TCPK) are used to substantiate the key function.

$$C_i = E_{\mathrm{AES}}\left(K_j, B_i\right) \tag{3}$$

where E_{AES} is the AES encryption function.

Phase II is implemented in conjunction with Phase I to raise the degree of security without raising the execution period. The remaining $^{n}/_{2}$ chains in Phase II are secured by the XOR-DUAL RSA method. The DUAL RSA method allows maximum encrypting and decrypting and is four times faster than the standard RSA technique. The XOR Encryption algorithm is a symmetric method of encryption and decryption that uses the same key. The XOR-DUALRSA algorithm ensures the creation of a more trusted algorithm shown as follows:

$$M_i = \sum_{i=n/2}^{i=n-1} (B_i) n/2 \le i \le n-1\ldots. \tag{4}$$

In this algorithm, two large prime numbers, viz. p and q, have been selected randomly. Here, $x = p \times q$, and the function $\varphi(x) = (p-1) \times (q-1)$. A number comparatively prime to φ is chosen; d and e are considered so that $e \times d = 1$ MOD $\varphi(x)$, yet, the PBK (e,x) is used for the encryption.

$$R_i = (Bi)^e \bmod x. \tag{5}$$

ASCII for (B_i) is received and converted into binary values

$$L_i = \mathrm{ASCII}(B_i) \tag{6}$$

where L_i is the utility for transferring message block to ASCII. R_i is ciphered text by using DUAL RSA.

$$C_i = (R_i)\mathrm{XOR}(L_i). \tag{7}$$

MD5 is applied to the ciphertext C_i. The optimized performance of the hashing function is security.

$$d_i = \mathrm{MD5}(c_i) \tag{8}$$

$$D_i = \mathrm{MD5}(C_i). \tag{9}$$

During the last stage of the encryption process, consistent hash values (d_i and D_i) along with a size of 128 bits are concatenated, and two $^{n}/_{2}$ blocks are combined to produce n block ciphertext. The encryption process is described in the algorithm.

$$Q = c_i + C_i \tag{10}$$

$$H = d_i + D_i. \tag{11}$$

5.1.1 Algorithm of 2-TCA

```
Input:   C_D (Cloud Data), S_K (AES Secret KEY), T(128-bit size of
         the block)
Output:  E_D (Encrypted Data), A_D( AES/ECC Hybrid Encrypted
         Data), DR_D ( XOR dual RSA Encrypted Data), H (Hash
         value for the Encrypted Data)

Step 1   Divide the data into defined 128-bit blocks, $b = C_D/T$;
Step 2   Assign loop threshold $i = 0$;
Step 3   **If** $i < b/2$, execute the following steps. **Else** go to **Step 14**;
Step 4   Starting with the first block of the cloud data,
           $m_i = \sum_{i=0}^{i=b/2}(Bi)$;
Step 5   Assign loop threshold $j = 0$;
Step 6   **If** $j < b - 1$, execute the following steps. **Else** go to **Step
           10**;
Step 7   $K_j = \mathrm{ECC}_{\mathrm{enc}}\ (TC_{C_D K}, S_{ki-1})$;
Step 8   Increment loop threshold, $j = j + 1$;
Step 9   **GOTO Step 6**;
Step 10  AES/ECC hybrid encrypted data, $A_d = E_{AES}(K_j,B_i)$;
Step 11  Apply an HF over the encrypted data, $d_i = \mathrm{MD5}(A_d)$;
Step 12  Increment loop threshold, $i = i + 1$;
Step 13  **GOTO Step 3**;
Step 14  Assign loop threshold $i = {}^{b}/_{2}$;
Step 15  For public-key encryption, choose two large prime
           numbers, $p$, $q$;
Step 16  $x = p*q$;
Step 17  $\varphi(x) = (p - 1) \times (q - 1)$
Step 18  The prime value is relatively chosen $\varphi$, $r$;
Step 19  Calculate $e \times r = 1$ MOD $\varphi(x)$;
Step 20  DUAL RSA encryption uses $(e, x)$ as the PBK
Step 21  **If** $i < b$, execute the following steps. **Else** GOTO **Step 14**;
Step 22  Starting with the second block of the cloud data,
           $M_i = \sum_{i=n/2}^{i=n}(B_i)$;
Step 23  $R_i = (Bi)^e \mathrm{MODx}$;
Step 24  $L_i = \mathrm{ASCII}(B_i)$, ASCII for $(B_i)$ is got and converted to
           binary
Step 25  $DR_i = (R_i)\mathrm{XOR}(L_i)$;
Step 26  Apply an HF over the encrypted data, $D_i = \mathrm{MD5}\ (DR_d)$;
Step 27  Increment loop threshold, $i = i + 1$;
Step 28  **GOTO Step 21**;
```

Input:	C_D (Cloud Data), S_K (AES Secret KEY), T(128-bit size of the block)
Output:	E_D (Encrypted Data), A_D(AES/ECC Hybrid Encrypted Data), DR_D (XOR dual RSA Encrypted Data), H (Hash value for the Encrypted Data)
Step 29	The two $^n/_2$ blocks are incorporated to create ciphertext of n blocks, $E_D = A_d + DR_d$;
Step 30	The corresponding hash values ($d_i$and D_i) with the size of 128 bits for every data is concatenated, $H = d_i + D_i$;

5.2 Decryption method

In decryption, ciphertext Q is separated into n blocks; it consists of 128 bytes, and then again, it is divided into two subsections, $c_i(0{:}^n/_2 - 1)$ and $C_i(^n/_2{:}n - 1)$ blocks. Hashing is used to determine whether or not the recipient gets some text in cipher. In both the encryption and decryption of data, the hash algorithm values can be compared. If they are equivalent, then the algorithm precedes the cycle of decryption. Otherwise, it discards the message that the first $^n/_2$ blocks have been decrypted by using ECC and AES algorithms, while hash values are equal at the source and sink nodes as follows:

$$c_i = \sum_{i=0}^{i=n/2-1} (Bi) 0 \le i \le n/2 - 1 \tag{12}$$

$$k_i = \mathrm{ECC_{dec}}\left(\mathrm{TC_{PK}}, K_{j-1}\right) 0 \le i \le n/2 - 1 \tag{13}$$

$$0 \le j \le L - 1.$$

The key of AES k_j along with bits length the ECC decrypts L for generating that is in used DAES (AES decryption function) for decrypting cipher document.

$$m_i = D_{\mathrm{AES}\left(K_j, c_i\right)}. \tag{14}$$

M_i is at the first portion of plain text, and the remaining $^n/_2$blocks have decrypted by using the following XNOR-DUAL RSA process:

$$C_i = \sum_{i=n/2}^{i=n-1} (Bi) n/2 \le i \le n - 1 \ldots \tag{15}$$

Private Key (d,p,q) is used for decrypting the data. For decryption process, various parameters are computed d_{p}- = d MOD (p − 1), $d_q = d$ MOD (q − 1), $R_{pi} = R_{idp}$ MOD p, $R_{qi} = R_{idq}$ MOD q,

$$S_0 = \left(R_{qi} - C_{pi}\right) p^{-1} \text{ MOD } q \tag{16}$$

$$S_i = R_{pi} + S_0 P. \tag{17}$$

ASCII for (C_i) is converted to binary

$$W_i = \mathrm{ASCII}(C_i) \tag{18}$$

where $L_i$is a function used for converting the block of ciphertext to ASCII.

$$M_i = S_i \mathrm{XNOR}\ W_i. \tag{19}$$

M_i is the second, plain text portion. The two $^n/_2$ blocks are inserted into the decryption cycle's final stage to generate plain text of n pieces.

$$P = m_i + M_i. \tag{20}$$

5.2.1 Decryption algorithm

Input:	E_D (Encrypted Data), H (Hash value for the encrypted data), T (128-bit size of the block), L(Key Length), d_i, D_i, K (encrypted key using ECC)
Output:	C_D (Cloud Data);
Step 1	Divide the encrypted data into defined 128-bit blocks, $b = E_D/T$;
Step 2	Assign loop threshold $i = 0$;
Step 3	**If** $i < {}^b/_2$, execute the following steps. **Else** go to **Step 15**;
Step 4	Starting with the first block of the encrypted data, $m_i = \sum_{i=0}^{i=b/2}(Bi)$;
Step 5	$\tilde{d}_i = MD5(A_d)$
Step 6	$\tilde{D}_i = MD5(DR_d)$
Step 7	Compare the hash values generated with the ones received **If** ($d_i = \tilde{d}_i$) and ($D_i = \tilde{D}_i$)
Step 8	Assign loop threshold j = 0;
Step 9	**If** j < L-1, execute the following steps **Else** GOTO **Step 13**;
Step 10	$K_j = \mathrm{ECC_{dec}}\ (TC_{C_D K}, S_{ki-1})$;
Step 11	Increment loop threshold, $j = j + 1$;
Step 12	**GOTO Step 9**;
Step 13	Increment loop threshold, $i = i + 1$;
Step 14	**GOTO Step 3**;
Step 15	Assign loop threshold $i = {}^b/_2$;
Step 16	Obtain the PK values, d, p, q;
Step 17	Measure the HFs: $d_p = d\mathrm{MOD}(p - 1)$ $D_p = d\mathrm{MOD}(q - 1)$
Step 18	**If** i < b, execute the following steps. **Else** go to **Step 24**
Step 19	Starting with the 2nd Block of the cloud data, $DR_{d_i} = \sum_{i=n/2}^{i=n}(B_i)$;
Step 20	$S_0 = \left(R_{qi} - C_{pi}\right)p^{-1}\mathrm{MOD}q$;
Step 21	$S_i = R_{pi} + S_0 P$;
Step 22	ASCII for (C_i) is converted to binary, $W_i = ASCII(C_i)$;
Step 23	**For** the second plain text portion, calculate: $M_i = S_i XNORW_i$;
Step 24	Increment loop threshold, $i = i + 1$;

Input:	E_D (Encrypted Data), H (Hash value for the encrypted data), T (128-bit size of the block), L(Key Length), d_i, D_i, K (encrypted key using ECC)
Step 25	**GOTO Step 18**;
Step 26	The two $^n/_2$ blocks are inserted into the final stage of the decryption cycle to generate plain text of n pieces,$P = m_i + M_i$;

Algorithm	Elliptic curve Diffie–Hellman
Step 1	Users A and B agree on a standard shared key for encryption
Step 2	Users A and B calculate their respective PBK and PKs
Step 3	Users A and B interchange their respective PBKs
Step 4	Users A and B possess PKs along with other users' PBKs

6 Secure multipath key exchange

6.1 Elliptic curve Diffie–Hellman (ECDH) key exchange protocol

The ECDH primary protocol is implemented on the discrete logarithm problem of the Elliptical Curve, which is faster than the nonlinear logarithm of the same security measure. A generates the Private Key (PK) nA, and the PBK = nA*G, where G represents the elliptical curve generator if users A and B select their secret key, which is used in private encryption. A public channel transfers PA to B. Using the same method (Fig. 3), B converts PB to A.

After obtaining the message from B, A chooses a specific hidden key KAB = nA*PB = nA*nB*G, which is the point on the specified elliptic curve.

6.2 Key Exchange Model

Four distinct structures, including the Certificates Authority (CA), customers, multi-owners, and Cloud Service Providers (CSP), are associated with the research investigation. The following are described in the competitive phase:

a. CA

CA is a reputable organization that provides credentials for the general public. The credential is a crucial component of safe contact, which serves a critical function in the main public networks. This usually contains the secret key, the name, the expiry date, and other confidential material.

b. Multi-Owners

The word multi-owners implies that several individuals own identical records. Each owner may change the data/file at any time. They also have licenses to read, compose, and

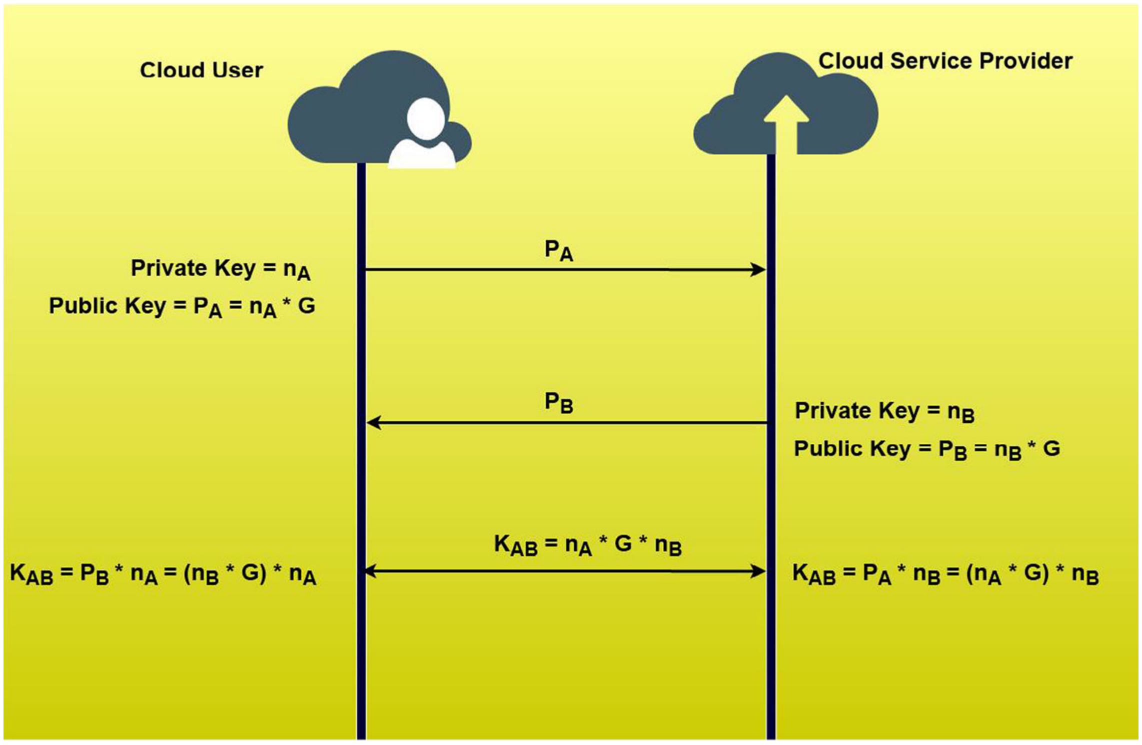
Cloud User
Cloud Service Provider
Private Key = n_A
Public Key = $P_A = n_A * G$
P_A
P_B
Private Key = n_B
Public Key = $P_B = n_B * G$
$K_{AB} = n_A * G * n_B$
$K_{AB} = P_B * n_A = (n_B * G) * n_A$
$K_{AB} = P_A * n_B = (n_A * G) * n_B$

Fig. 3 Sequence diagram for MECDH key exchange protocol

delete. The owner may handle all (or part) of the records, i.e., reading, writing, editing, and storage. The primary proprietor is responsible for initializing the device and authorizing the user. When the primary owner validates with CA, the primary owner can connect other owners to the cloud and grant approvals for entry, such as reading, writing, and alteration. Our suggested modified elliptic curve Diffie–Hellman (MECDH) algorithm uses a file encryption technique for getting each file submitted with the Access Control List (ACL) to the server to avoid the MITM attack.

c. Users

Users are organizations that can quickly interact with various cloud services and proprietary software. They can update and decrypt encrypted data from the cloud storage service if their characteristics meet the access control policy. The client is a specified individual who can read the file but not write or delete it.

d. CSP

The CSP offers enormous support for the clients. This helps the owners to build, upload, and exchange data with other owners and users (documents, folders, photos, videos, etc.).

6.3 Overview of proposed MECDH algorithm

- MECDH indicates workflow in Fig. 4.
- The primary owner of the data (O1) signs with CA.
- After the primary owner (O1) has been licensed with CA, it includes a PBK certificate. After completing the credential, the primary owner may connect other users to the cloud and access permissions (such as reading, editing, and writing) to the data via ACL. Owners can also connect with users that require learning.
- CA authorization of owners uses passwords (Email ID, User Name) to exchange data with the other users and proprietors.
- If two connection holders attempt to authenticate each other, they must trigger mutual authentication.
- For example, the shareholders (O2, O3, given two shareholders) seek entry from the primary Owner (O1). Using the MECDH algorithm, the primary exchange value is determined. Both proprietors build and validate the signature using the ECDS algorithm to stop the MITM attack.
- Owners share a shared hidden authentication key. This approach lets the owners authenticate each other, and anonymity is protected while exchanging cloud data.
- The proprietors encrypt and transfer the data/file to the server using the MECDH.
- CSP proprietors/users request data/files to access the data owners' policies for expected users. Next, the users open the ciphertext. The user submits the decryption key request to the principal owner to decrypt the data/files.
- If owners/users require a cloud connection to data, the CSP must verify whether the owner or customer is an approved entity or not. Once owners or customers have been approved, they may again access cloud data. Users

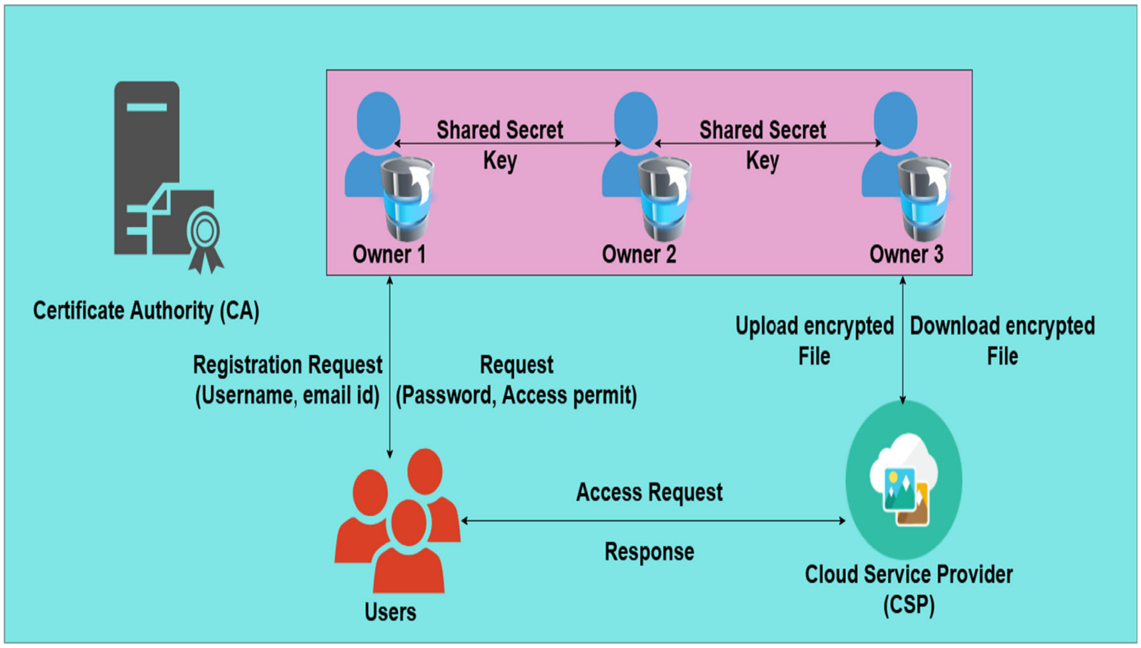
Shared Secret Key
Shared Secret Key
Owner 1
Owner 2
Owner 3
Certificate Authority (CA)
Upload encrypted File
Download encrypted File
Registration Request (Username, email id)
Request (Password, Access permit)
Access Request
Response
Users
Cloud Service Provider (CSP)

Fig. 4 MECDH data are sharing between users and owners

would be able to retrieve the cloud's ciphertext if their attributes match the owners' defined access tree.

6.4 Multi-owner mutual authentication using MECDH

The central agreement on the EMCDH is not restricted to agreeing to a main that only two parties share. Any number of users/owners participate in an organization by conducting contract protocol iterations and intermediary data exchange. For examples, Owner 1, Owner 2, and Owner 3 (O1, O2, and O3) may engage in Diffie–Hellman deal and measure the mutual secret keys (R1, R2, and R3) as follows:

1. The session's symmetric key are the parties that agree on the highest first number *'p'* and points on the elliptic curve P and K.
2. The parties create PKs, called n_A, n_B, and n_C.
3. O_1 computes $R_1 = n_A*P\%p$ and transfer it to O_2 and O_3.
4. O_2 computes $R_2 = n_B*P\%p$ and transfer it to O_3 and O_1.
5. O_3 computes $R_3 = n_C*P\%p$ and transfer it to O_1 and O_2.
6. O_1 computes $K = n_A*R_2*R_3P\%p$.
7. O_1 computes $K = n_A*n_B*n_CP\%p$.
8. O_2 computes $K = n_B*R_1*R_3P\%p$.
9. O_2 computes $K = n_A*n_B*n_CP\%p$.
10. O_3 computes $K = n_C*R_1*R_2P\%p$.
11. O_3 computes $K = n_A*n_B*n_CP\%p$.

6.4.1 Authentication and mutual key interchange appliance

As shown in Fig. 5, the owners (O1, O2, and O3) use their passwords (Email ID, User Name) to authenticate with CA. It verifies the credentials and then reacts with the PBK certificate of owners when credentials are legitimate. Therefore, they would not grant the certificates. To stop the MITM attack, the transmitting owners test themselves for reciprocal security utilizing the Diffie–Hellman key exchange before exchanging their keys, determining their common key values. This approach lets the owners authenticate each other, and anonymity is protected while exchanging cloud data.

7 Experimental environments

Cloud Instances (CI's) are configured as needed. Every CI has one hyperthreaded CPU core (4.0 GHz turbo boost frequency), 4 GB RAM, and a 20 GB local SSD disk. The cloud example runs CentOS 7 (minimum version) to use fewer approaches from other systems, and the software examples are configured and managed by the front-end Web server. The CFE server has a simple load balancer that operates in a simplistic round-robin fashion.

The Cloud Instances log files the client's execution time (if any), request, and reply for plaintext, TLS *v1.2,* and *v1.3*, and the DiTD, with and without the availability of the user to connect to the cloud service. The Cloud User (CU) thus tracks round-trip time details for more review on the client-side. All CUs run iteratively, and each time submits requests with a different size (100 KB, 500 KB, 1 KB, 500 KB, or 1 MB) of the data. A dedicated protected system for the authentication and delivery of shared keys acts as a Key Management Server (KMS) to handle collective root keys. In KMS, all cloud organizations are registered with their parent PBKs in contradiction to the unique identifier. In DiTD, the server CFE and CUs file the cryptographic keys with their IP addresses and allocate tokens with random strings. Both tests are carried out in iterative form (1000-fold). That request is appropriate to short encrypted sessions with the hashed session key created from the property of the link and details received by the client.

8 Results and discussion

This segment discusses the findings, and the solution is analyzed. The standard cryptographic methods (digital signature, PUKC and authentication, cryptographic hash, and symmetric block encryption) are iteratively tested for various payload sizes (100 to 200 MB) for choosing the best option for implementing high-performance cloud-based secured protocol (i.e., DiTD), which utilizes such technologies effectively concerning their length. The next segment provides a detailed DiTD vulnerability review toward multiple forms of threats. Before that, we analyze DiTD's output in terms of server-side execution time, plaintext bandwidth overhead, client-side round-trip time, server-side memory consumption, and the effect of different payload sizes in the scenarios mentioned above.

8.1 Server-Side Execution Time

Figure 6 shows the average time in *ms* for the public clouds. It examines average execution times in various

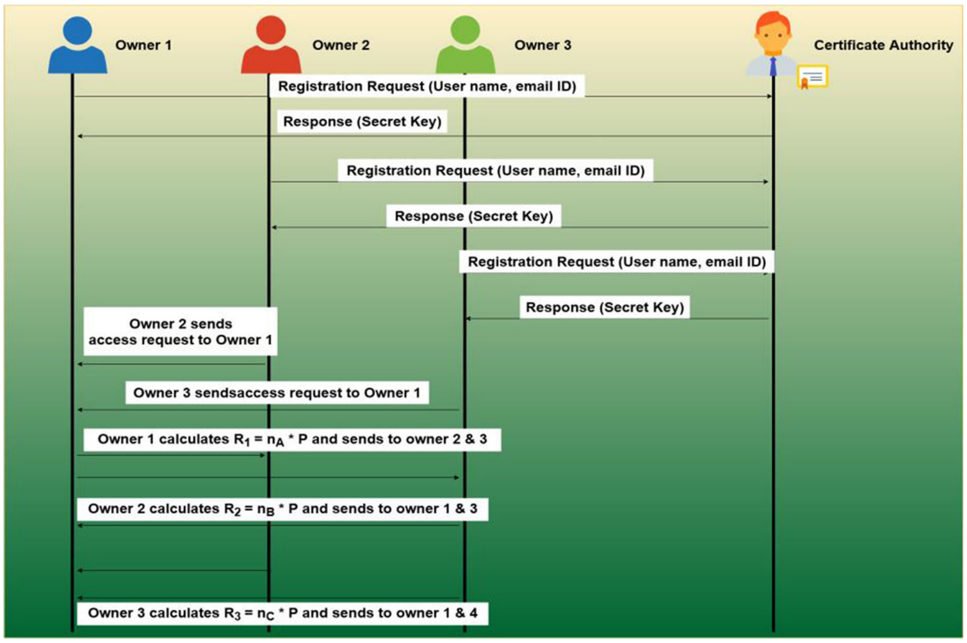
Owner 1
Owner 2
Owner 3
Certificate Authority
Registration Request (User name, email ID)
Response (Secret Key)
Registration Request (User name, email ID)
Response (Secret Key)
Registration Request (User name, email ID)
Response (Secret Key)
Owner 2 sends access request to Owner 1
Owner 3 sendsaccess request to Owner 1
Owner 1 calculates $R_1 = n_A * P$ and sends to owner 2 & 3
Owner 2 calculates $R_2 = n_B * P$ and sends to owner 1 & 3
Owner 3 calculates $R_3 = n_C * P$ and sends to owner 1 & 4

Fig. 5 Owner authentication and key exchange

cloud examples such as TLS *v1.3*, TLS *v1.2*, and DiTD for different payload sizes (Ex., 100 KB, and 500 KB; 1 KB, 500 KB, and 1 MB). DiTD mechanism outperforms TLS *v1.3* considerably for all payload sizes. DiTD performs about 90% faster than TLS *v1.3* communication. The proposed model produces better outcomes with the DiTD mechanism, which depends upon TLS *v1.2*.

8.2 Client-side round-trip time

On the client side, it measures average round-trip time (in *ms*) using the sum of experimental durations for the session establishment, the connection creation (if it occurs present), request-response time of different payload sizes. Figure 7 represents the average round-trip time for examined client instances under plaintext, TLS *v1.2*, TLS *v1.3*, and DiTD for different payload sizes (100 KB, 500 KB, 1 KB, 500 KB, and 1 MB). While observing the client-side average round-trip time's performance curves, the DiTD mechanism shows promising performance against TLS *v1.2* and TLS *v1.3*.

8.3 Bandwidth overhead

The bandwidth of the graph shown in Fig. 8 is considered based upon the bandwidth utilization of plaintext communication. By this, it immediately notifies the bandwidth overhead of the 100 bytes based upon the payload size, which is higher than 28% for TLS *v1.3*. Fortunately, for DiTD, the overall process with plaintext communication alone displays 80% and provides a 54% advance over the TLS *v1.3*.

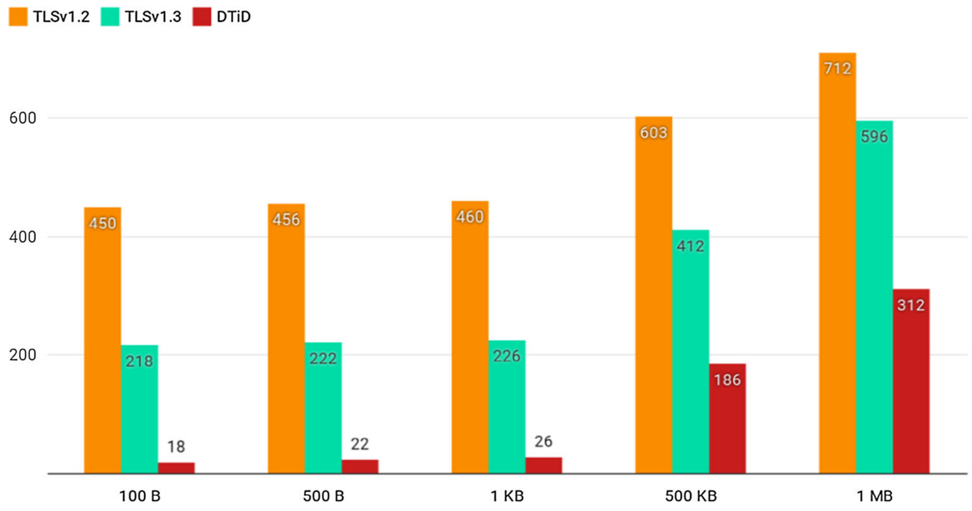

Fig. 6 Server-side execution time

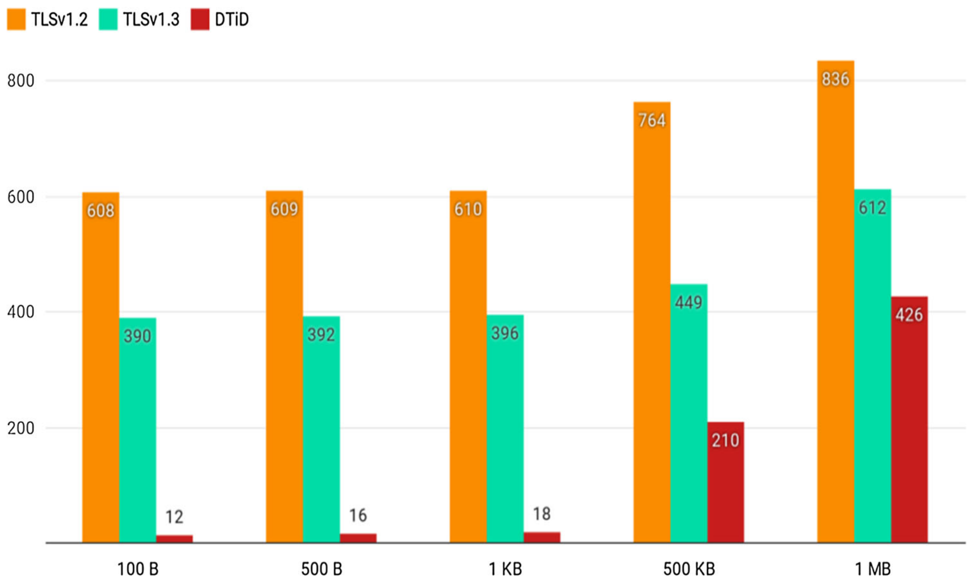

Fig. 7 Client-side execution time

For 1 KB of payload size, the DiTD mechanism delivers 32% increases over bandwidth utilization of TLS *v1.3*. This graph displays diminishing payload sizes, and increases for payload size overhead (500 KB) turn into nearly 1% in all the types of communications according to the plaintext. Hence, the large volume of data is negligible in an overhead process—however, DiTD is implemented prominently well in lesser payload sizes and greater payload sizes.

8.4 Server-side memory consumption

Figure 9 represents server-side memory usage of DiTD considered in cloud instances depending upon the plaintext, TLS *v1.3*, and TLS *v1.2* communications. Figure 9 shows the DiTD mechanism, which displays the memory usage with a reasonable quantity of dissimilar payload sizes that lie much closer to TLS *v1.2* and TLS *v1.3* communications. And the pattern usage images have similar behavior, which increases proportionately with increasing payload size in all scanned cloud cases and memory usage.

Overall, the DiTD mechanism performs significantly well than TLS *v1.3* at the server-side performance, bandwidth overhead, memory usage at server side, and client-side round-trip time.

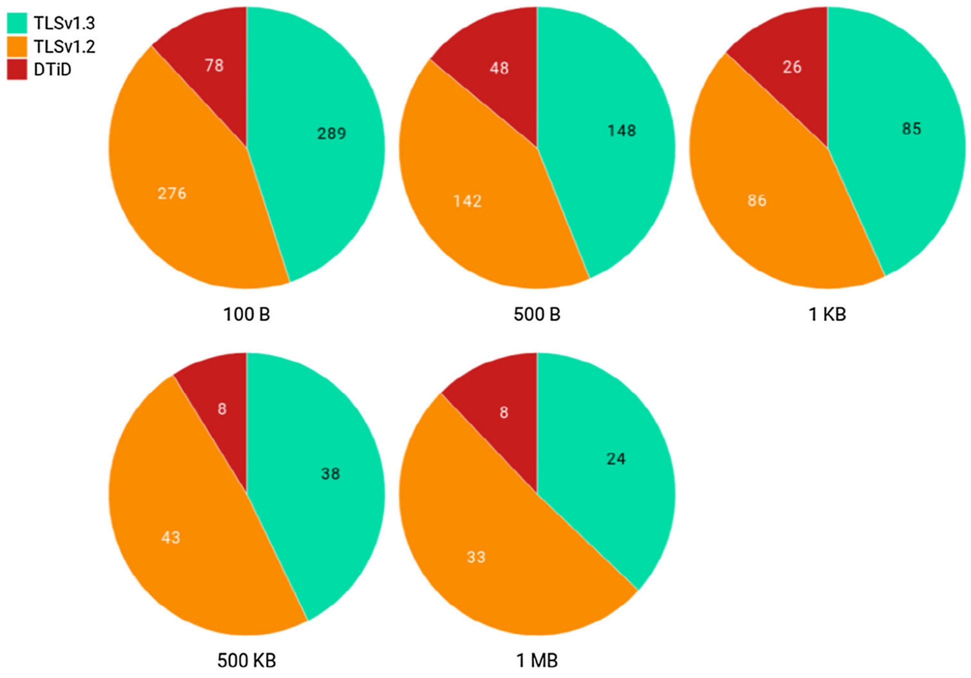

Fig. 8 Bandwidth while overhead occurs

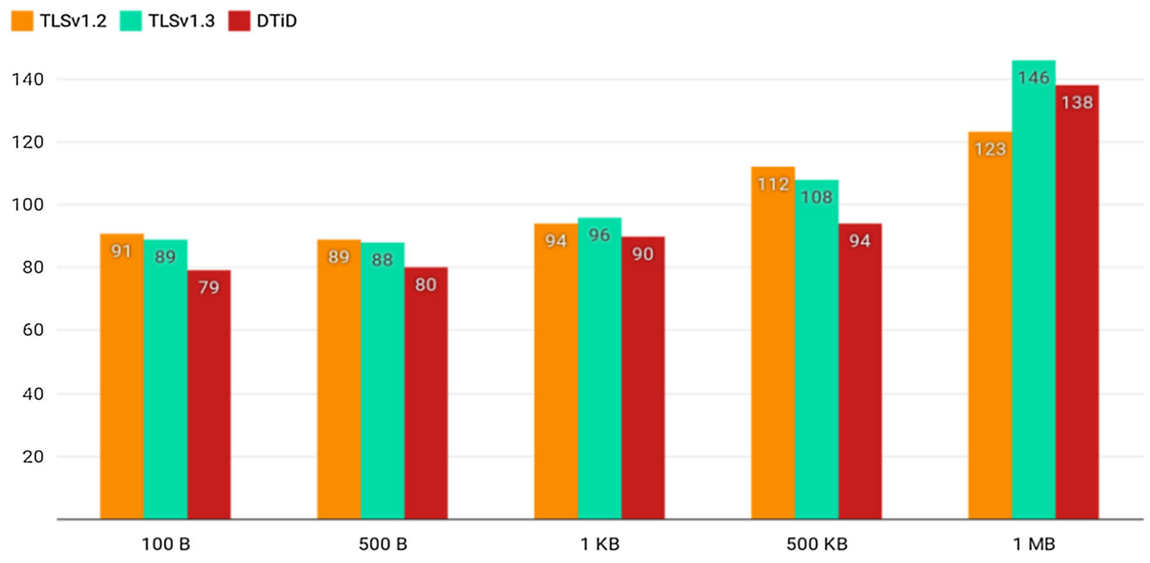

Fig. 9 Server-side memory consumption

A 256-bit encrypted channel is conclusively recognized during the session development phase without affecting the overhead performance, memory, or bandwidth.

9 Conclusion

In this paper, we have suggested a comprehensive, secure architecture for cloud communication. In DiTD, the security in data-in-transit and the legitimacy of the cloud entities have guaranteed and determinedly incorporated into communications for protecting, in contrast to the full range of the cloud assaults. A unique high-performance cloud dedicated to the 2-TCA algorithm is intended and also applied. It takes seven highly closed new message organizations that establish more secured performance and the bandwidth-efficient protocol along with practical memory usage. It is highly successful and secure Man-In-The-Middle (MITM) (with identity spoofing, eavesdropping, sniffing, data intrusive), replay, compromised-key, sensitive information disclosure, session hijacking attacks, and repudiation. DiTD produces 90% execution time, which is more than that of the TLS *v1.3* (latest version between SSL successors) on the server side, and it shows similar performance at the client-side. It produces 54% more than the TLS v1.3 and overall reasonable memory usage, contrary to the dissimilar payload sizes in bandwidth consumption. As a basis of the security aimed at the data transit in cloud computing, it enforces the NIST recommendation.

Funding Not Applicable.

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

Adrian, D., Bhargavan, K., Durumeric, Z., et al. (2015) Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. pp 5–17. CCS'15, ACM, New York, NY, USA. https://doi.org/10.1145/2810103.2813707.

Almogren A (2019) An automated and intelligent Parkinson disease monitoring system using wearable computing and cloud technology. Clust Comput 22(1):2309–2316

Aviram, N., Schinzel, S., Somorovsky, J., et al. (2016) Drown: Breaking TLS using SSLv2. In: USENIX Security Symposium. pp 689–706

Bittau A, Hamburg M, Handley M et al. (2010)The case for ubiquitous transport-level encryption. In USENIX Security Symposium, pp 403–418,

Cangialosi F, Chung T, Choffnes D et al. (2016) Measurement and analysis of private key sharing in the HTTPS ecosystem. In Proceedings of the 2016 ACMSIGSAC Conference on Computer and Communications Security, pp 628–640, New York, NY, USA. ACM.

Cramer R, Shoup V (2004) Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J Comput 33(1):167–226

Elazhary H (2019) Internet of Things (IoT) mobile cloud cloudlet mobile IoT cloud fog mobile edge and edge emerging computing paradigms: Disambiguation and research directions. J Netw Comput Appl 128:105–140

Flavel A, Mani P, Maltz D et al. (2015) Fast route: A scalable load-aware anycast routing architecture for modern cdns. In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp 381–394, Oakland, CA. USENIX Association.

Gawannavar M, Mandulkar P, Thandeeswaran R, Jeyanthi N (2015) Office in cloud: approach to authentication and authorization. Recent Adv Commun Netw Technol Bentham Sci 4(1):49–55

Hamad SA, Sheng QZ, Zhang WE et al. (2020) Realizing an Internet of Secure Things: A Survey on Issues and Enabling Technologies, In IEEE Communications Surveys and Tutorials, vol. 22, no. 2, pp. 1372–1391, Second quarter.

Jurcut A, Niculcea T, Ranaweera P et al. (2020) Security considerations for internet of things: a survey. SN Comput SCI 1:193

Kodali, R., Sarma, N., (2013) Energy-efficient ECC encryption using ECDH. In: Emerging Research in Electronics, Computer Science and Technology, Lecture Notes in Electrical Engineering, Vol. 248. Springer, pp 471–478.

Liu Y, Tome W, Zhang L et al. (2015) An end-to-end measurement of certificate revocation in the web's PKI. In Proceedings of the 2015 Internet Measurement Conference, pp:183–196, New York, NY, USA. ACM.

Megouache L, Zitouni A, Djoudi M (2020) Ensuring user authentication and data integrity in multi-cloud environment. Hum Cent Comput Inf Sci 10:15

Mohiuddin I and Almogren A (2020) Security Challenges and Strategies for the IoT in Cloud Computing, 2020 11th International Conference on Information and Communication Systems (ICICS), pp 367–372.

Shailendra R, Arun KS, Park JH (2018) A novel framework for internet of knowledge protection in social networking services. J Comput Sci 26:55–65

Singh HJS and Khanna MS (2020) Cloud's Transformative Involvement in Managing Big-Data Analytics for Securing Data in Transit, Storage And Use: A Study, 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), pp 297–302.

Tillich, S., Großschädl, J., 2005. Accelerating AES using instruction set extensions for Elliptic Curve cryptography. In: Computational Science and its Applications–ICCSA, vol. 3481., pp 665–675.

Veerabathiran VK, Mani D, Kuppusamy S et al (2020) Improving secured ID-based authentication for cloud computing through novel hybrid fuzzy-based homomorphic proxy re-encryption. Soft Comput 24:18893–18908

Vijaya Kumar V, Devi M, Vishnu Raja P, Kanmani P, Priya V (2020) Sengan Sudhakar, Krishnamoorthy Sujatha, Design of peer-to-peer protocol with sensible and secure IoT communication for future internet architecture. Microprocess Microsyst 78:103216

Authors and Affiliations

Keerthana Nandakumar[1] · Viji Vinod[2] · Syed Musthafa Akbar Batcha[3] · Dilip Kumar Sharma[4] · Mohanraj Elangovan[5] · Anjana Poonia[6] · Suresh Mudlappa Basavaraju[7] · Sanwta Ram Dogiwal[8] · Pankaj Dadheech[9] · Sudhakar Sengan[10]

✉ Keerthana Nandakumar
keerthana.mca@drmgrdu.ac.in

Viji Vinod
vijivino@gmail.com

Syed Musthafa Akbar Batcha
syedmusthafait@gmail.com

Dilip Kumar Sharma
dilipsharmajiet@gmail.com

Mohanraj Elangovan
csemohanraj@gmail.com

Anjana Poonia
anjanapoonia.ap@gmail.com

Suresh Mudlappa Basavaraju
sureshresearch45@gmail.com

Sanwta Ram Dogiwal
dogiwal@gmail.com

Pankaj Dadheech
pankajdadheech777@gmail.com

Sudhakar Sengan
sudhasengan@gmail.com

[1] Department of Computer Science, Dr. M.G.R Educational and Research Institute, Chennai, Tamil Nadu 600095, India

[2] Department of Computer Applications, Dr. M.G.R Educational and Research Institute, Chennai, Tamil Nadu 600095, India

[3] Department of Information Technology, M. Kumarasamy College of Engineering, Karur, Tamil Nadu 639113, India

[4] Department of Mathematics, Jaypee University of Engineering and Technology, Guna, Madhya Pradesh 473226, India

[5] Department of Computer Science and Engineering, K. S. Rangasamy College of Technology, Tiruchengode, Tamil Nadu 637215, India

[6] Department of Master of Computer Application, Sri Balaji College of Engineering & Technology, Jaipur, Rajasthan 302013, India

[7] Department of Information Science and Engineering, East West Institute of Technology, Bengaluru, Karnataka 560091, India

[8] Department of Information Technology, Management & Gramothan (SKIT), Swami Keshvanand Institute of Technology, Jaipur, Rajasthan 302017, India

[9] Department of Computer Science and Engineering, Management & Gramothan (SKIT), Swami Keshvanand Institute of Technology, Jaipur, Rajasthan 302017, India

[10] Department of Computer Science and Engineering, PSN College of Engineering and Technology, Tirunelveli, Tamil Nadu 627152, India