TMS – A Trusted Mail Repository Service using Public Storage Clouds

João Rodrigues
FCT/UNL – CITI*
Campus da Caparica
2825-465 Caparica
jm.rodrigues@campus.fct.unl.pt

Bernardo Ferreira
FCT/UNL – CITI*
Campus da Caparica
2825-465 Caparica
bernardof@acm.org

Henrique Domingos
FCT/UNL – CITI*
Campus da Caparica
2825-465 Caparica
hj@fct.unl.pt

ABSTRACT

In this paper we present the Trusted Mail System (TMS), a dependable Email repository service that explores multiple untrusted storage clouds for storing, accessing and searching private email data. The system architecture provides security and reliability services while leveraging the heterogeneity and diversity offered by different untrusted cloud storage solutions from different service providers. To address dependability issues, TMS enforces a security model that protects confidentiality and integrity of mailboxes stored in those clouds, adding availability, reliability and intrusion-tolerance guarantees. The system uses homomorphic encryption mechanisms and indexing techniques allowing ranked multi-keyword searching operations over encrypted email messages and its contents. We illustrate TMS feasibility from an implemented prototype, evaluating its performance, design options, and services. The experimental results show that the solution is viable, offers reliability and privacy control for the users and does not aggravate conditions of data-access latency and availability.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General-Security and protection; C.2.4 [**Distributed Systems**]: Distributed applications; D.4.5 [**Reliability**]: Fault-Tolerance

General Terms

Algorithms, Design, Performance, Reliability, Security

Keywords

Untrusted Cloud Storage; Threshold Signatures; Secret Sharing; Homomorphic Encryption; Email Security and Reliability; Searchable Encryption.

1. INTRODUCTION

Most companies consider email to be a mission critical application [1]. Considerable information related with intellectual property is processed via email services and applications. Likewise, many information leakage actions involves email related discovery and email messages are commonly used to support strategic commercial information or to confirm business transactions. As such, email repositories are examples of systems where reliability and security concerns must be carefully addressed. Despite this, email services for individual, enterprise or institutional use form one of the most popular contexts for data outsourcing on public Internet Cloud Storage Providers (ICSPs). Such solutions tend to be used as outsourced email repositories for Internet ubiquitous access, using Webmail or conventional Mail User Agent (MUA) applications running in different user devices. In these cases, it is common to observe a clear contradictory approach in the way how cloud-based email outsourcing services are adopted: in one hand, cloud storage services provide no dependability properties under the control of end users; on the other hand, many studies have rated security and privacy to be major areas of concern and obstacles to adopt cloud solutions [2].

In an attempt to improve the reliability, availability and security conditions of dependable cloud storage services, the use of multiple storage clouds offers an innovative, yet challenging research direction [3]. Such approach allows the materialization of a transparent and dependable cloud-of-clouds data repository architecture. These solutions benefit from the resilience conditions established by the diversity of multiple clouds, as well as from the security controls that can be provided by integrated cryptographic methods and data-replication techniques, under the control of end users, running in trusted computing bases (TCBs) [3]. It is also an interesting design option in addressing intrusion-tolerance, leveraging from hardware/software heterogeneity and independent failures/attacks in each individual cloud [4].

Inspired from the relevant work on dependability services in the design of cloud-of-clouds data-storage architectures [3, 4], this paper addresses the design and implementation of TMS (Trust Mail System), an email repository service based on a storage backend on top of a cloud of internet storage clouds, as repository components offered by current ICSPs. TMS offers security, privacy, availability and reliability guarantees for the mail repositories, under control of the users. The solution is designed to run as a middleware service, deployed as a local proxy (in a client machine) or as a trusted remote proxy (used as a trusted service). The system provides SMTP and POP3 standard operations for MUAs and WebMail applications, as well as an auxiliary API which translates read/write/search operations on mailboxes and contents, to the equivalent operations in the backend storage supported in a cloud-of-clouds architecture.

The main contribution of this work lies in the proposal of a novel architecture for email data outsourcing, enforcing:

- **Security and Privacy:** The proposed middleware assures the security and privacy of mailbox data by combining traditional

* Faculty of Sciences and Technology, New University of Lisbon, Center for Informatics and Information Technologies.

This work is supported by FCT-UNL PhD Fellowship Program and FCT-MEC under the PTDC/EIA/113729/2009 Grant (SITAN - Services for Intrusion Tolerance in Ad-Hoc Networks)

MW4NG '13 December 9-13, 2013, Beijing, China

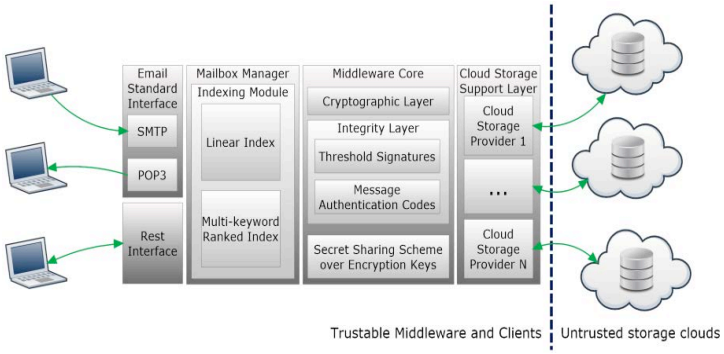

Figure 1 - *TMS Architecture and System components*

Operation	**Description**
ObjectId ***put*** *(Mail)*	Writes and stores on the clouds a new mail message through the TMS middleware
Mail ***get*** *(ObjectId)*	Retrieves a mail message from the storage clouds using it's identifier
Set<ObjectId> ***list*** *()*	Lists the mail message identifiers that compose the user's mailbox
Set<ObjectId> ***searchContent*** *(Set<Keyword>)*	Searches the user's mailbox and mail contents for a set of keywords, returning the set of relevant mail message ids
Set<ObjectId> ***searchMetadata*** *(Set<MetadataKeys>)*	Searches the user's mailbox for a set of mail metadata keys (ex: Sender="Alice" AND CC_Contains="Bob")

Table 1 - *TMS Rest API, providing complementary operations to the SMTP and POP3 standards and an auxiliary access point for clients not able to implement those standards*

cryptography, homomorphic encryption algorithms and state-of-the-art threshold mechanisms;

- **Provider Independent Availability:** The proposed middleware ensures email service availability by using multiple instances of the TMS architecture and multiple cloud repositories as storage support;
- **Confidentiality, Integrity and Authenticity without Key Management overhead:** In order to offer privacy, integrity and authenticity of the stored emails, TMS uses secret sharing and threshold based signature schemes, enabling the secure storage of cryptographic keys (used in the encryption mechanisms) in untrusted cloud storages and signatures over data without the need of managing verification keys;
- **Privacy Preserving Search Operations:** A popular functionality in email services is the ability to search over the user's mailboxes. TMS offers secure Boolean and ranked search operations over email data, by combining well established information retrieval techniques with state of the art homomorphic encryption algorithms;

2. TMS ARCHITECTURE AND MODELS

This section presents TMS architecture as a middleware layering solution and its main software components and models.

2.1 System Model and Definitions

In this paper, we define ***n*** as the total number of storage clouds used, ***f*** as the number of clouds that can be attacked or fail and ***t*** as the threshold number of clouds required for the Threshold Mechanisms employed (ex: validate a threshold signature or recover a secret, more details in section 3.1).

TMS's system model follows a design oriented to a cloud-of-storage clouds architecture, addressed by a middleware solution between Mail User Agent applications (MUAs) or Web-Mail based Application Servers and a set of multiple data-storage clouds, as currently provided by ICSPs. The system is designed and implemented as a software proxy service, allowing a data-access model supported by SMTP and POP3 endpoints (with possible support for SSL-based client interactions, formerly known as STARTTLS, defined in RFCs 2595 and 3207). Additionally, the TMS service is also available via a Web REST-based API, providing read/write/search operations over the mail messages (as defined in RFC 5322).

In the system model, applications using the TMS services implement the client side of SMTP and POP3 standards. Applications can also access the system through the REST API (Table 1) in order to write, read or search mail messages. Both access methods offer a transparent way of communication as with standard SMTP/POP3 servers and popular ICSPs. We consider an asynchronous distributed system setting for the TMS middleware. External applications act as writers, readers or searchers of mail messages in mailboxes. The middleware backend implements a transparent and uniform object access layer, supporting a replication process in which mail messages are replicated while encrypted through connectors to different ICSPs. In this backend level, reads and writes are supported as operations provided by the used backend clouds. Read operations can fail with a subjacent arbitrary failure model and write operations (for replicas of the same value) can arbitrarily fail at most ***f*** times, as long as ***t=2f+1*** writes of the same replica are correct (beyond this restriction only fail-stop faults are supported).

Figure 1 presents the TMS architecture and system components. Additionally to the referred APIs and backend cloud Connectors, different components in the middleware core implement the cryptographic algorithms and indexing mechanisms required to support the TMS functionalities. These mechanisms and their justification are discussed in more detail in section 3.

2.2 Security Concerns and Adversary Model

In the system model, the trust base for preserving conditions of dependability, availability, security and privacy is restricted to the components of the TMS middleware system. The storage clouds are considered not trustable, admitting they may be subject to both active attacks on the clouds' infrastructure (done by External Hackers and possible accidental/careless maneuver by ICSPs employees), as well as passive attacks done from inside the cloud servers (the *Malicious System Administrator* and *Curious but Honest* cloud models, as has been described in state-of-the-art related works [3, 4, 10]). To support reliability and intrusion tolerance of up to ***f*** faulty (or attacked) clouds, we adopt a set of ***n=3f+1*** untrusted storage clouds and ***t=n-f*** threshold/secret shares, with ***n*** being the same for both the storage of mail replicas and generated threshold shares. The rational for this is that if we minimize the number of shares distributed through the different clouds, we reduce the security levels of the secret sharing and threshold signature mechanisms. On the other hand, increasing the number of shares requires the employment of more independent clouds. For specific scenarios, this tradeoff may be addressed by specific parameterization. However, for implementation purposes, we decided to have ***t=n-f***, while also leaving room for a possible future employment of a byzantine fault-tolerant protocol requiring ***3f+1*** replicas to decide on a correct value.

The local servers where the TMS system is deployed are controlled by the clients and are considered a secure computing base. As an additional measure, and although the TMS servers are considered trusted, it is convenient to consider the possibility of an intrusion, aiming at breaking the privacy of locally stored data,

when TMS is not processing it's external operations over SSL sessions. As such, we minimize data exposure by keeping references and indexes encrypted with message digests and homomorphic encryption schemes, respectively.

2.3 Back-End Storage and Data Model

The middleware data model is specified in two domains: the middleware domain, where mailbox indexes are stored; and cloud storage, where actual email data is stored. Figure 2 represents TMS's data model. As seen in the upper part of the Figure, the middleware keeps a local version of the users' mailboxes, containing pointers to the actual emails. Each of these user's mailbox is composed by three indexes:

- **Reference Index.** Co-relates message *ids* with tokens composed by: a cloud reference, pointing to the objects in the cloud repository; the cryptographic key used to encrypt the Cloud Object; and optionally a Message Authentication Code (based on secure hash-functions) for fast authenticity and integrity checks;
- **Multi-Keyword Ranked Index.** A homomorphically encrypted search index, which allows search operations over encrypted email message contents while preserving their privacy. A search in the index returns a set of unique message identifiers, translated to in-cloud references through the reference index;
- **Boolean Index.** Allows fast searching over email header fields of standard email message formats (defined in RFC 5322), including recipients, sender or subject;

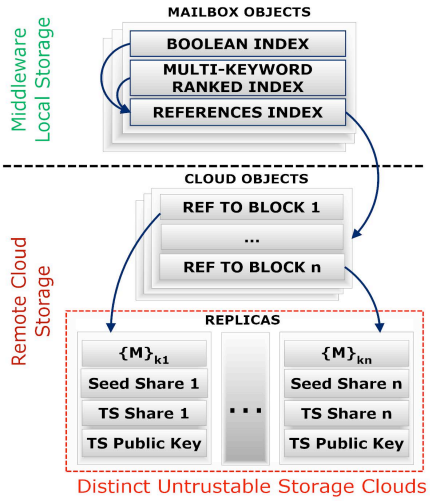

Figure 2 - *TMS Data Model and Back-end Storage, divided between TMS's local storage and cloud storage*

Both the ranked and Boolean indexes are built using a combination of homomorphic encryption algorithms (details in section 3.1) and information retrieval techniques (as discussed in [11], details omitted in this paper due to space restrictions). Additionally (although not yet implemented and validated in section 4) these structures can periodically be replicated to the clouds, using directly TMS's write operation to upload their state as mail messages. In the storage clouds (lower part of Figure 2), two data structures are required in the designed data model:

- **Cloud Object.** Represents an email message by referencing a set of data blocks. This object is encrypted using a PBE encryption scheme, where the password is protected and stored on the middleware reference index. Although the Cloud Object has an unique representation in the following Figure 2, it can be replicated alongside the data blocks, offering a higher level of availability;
- **Data Blocks.** A set of data blocks, representing each of the replicas of the email data. Each of these blocks stores encrypted data along with: a share of the seed used to generate the different cryptographic keys required in the solution; a share of the threshold signature used; and a public verification key for the threshold signature.

Both the data blocks and Cloud Object references are generated based on all object data and are used as unique identifiers in the key-value backend data storage clouds.

3. SYSTEM COMPONENTS

In this section we describe the mechanisms and algorithms that compose the different components of the TMS middleware core, as described in section 2 and represented in Figures 1 and 2.

3.1 Cryptographic Algorithms and Schemes

This section presents the most relevant cryptographic primitives used in TMS system model and architecture: Secret Sharing, Threshold Signatures and Homomorphic Encryption Schemes. More specific details on the security and performance of these schemes had to be omitted due to space limitations, but can be found in the respective references.

Secret Sharing Schemes. In our solution approach, we employ a secret sharing scheme [5] in order to safeguard TMS's cryptographic keys and securely store them in the untrusted storage clouds. This secret sharing scheme is used in a Byzantine set (see section 2.2), where ***t*** shares allow recovering a secret but ***t-1*** give no information to an attacker. The main advantage of this approach is the possibility of safeguarding cryptographic keys in a cloud-of-clouds distributed scenario, without requiring multiple backup copies or a Key Distribution Center (KDC). In the TMS architectural setting, three secret sharing schemes were implemented and experimentally evaluated: the **Shamir** [6], **Blakley** [7] and **Asmuth-Bloom** [8] schemes. The conducted evaluation revealed that the three schemes possess similar performances. As such, and considering that from the three it was the one with a most stable implementation, the Blakley Scheme was chosen as the secret sharing scheme to be used in the TMS prototype discussed and evaluated in section 4.

Threshold Signature Schemes. TMS employs Threshold Signatures [9] in order to obtain strong authenticity guarantees on its replication process. When compared to more conventional digital signature algorithms, this kind of signatures have the advantage of offering Byzantine consensus in an asynchronous environment while reducing the size of each signature and providing no correlation between signature shares. In such a scheme, a public key and at least ***t*** out of ***n*** shares are required to verify a signature. TMS's threshold signature implementation was adapted from [9] in such way that each signature and public key could be wrapped in raw data sets in order to be distributed evenly by multiple storage clouds. This scheme explores RSA along with Lagrange interpolation scheme, and so its security relies on the difficulty of RSA (discrete logarithm problem) and interpolation problems. Furthermore the proof of correctness of each share based on discrete logarithm problem avoids the poisoning of the signature verification process.

Homomorphic Encryption Schemes. In the context of TMS, the need to protect the privacy of search operations, minimize data exposure during a possible attack on the TMS infrastructure and, at the same time, a requirement for fast and efficient operations, led to the employment homomorphic encryption algorithms [10]

in the solution architecture. In particular, two partially homomorphic schemes were deployed: **Search Scheme** [11], a homomorphic encryption scheme designed for text data, allowing equality comparison between encrypted keywords of an email message or query. When used in combination with well-known information retrieval techniques, it can support privacy-preserving ranked searches over TMS mailboxes; **Paillier** [12], an asymmetric cryptographic algorithm, with additive homomorphic encryption properties. We point the reader to [11] and [12] for more details on the Paillier scheme and the security and efficiency of the algorithm. Through the application of this scheme it is possible to encrypt the metrics stored in TMS's indexes, while preserving the ability to perform searches over them.

3.2 System Processing and Algorithms

This section describes in detail the different steps required to complete the most relevant operations in TMS's APIs, the *put* and *get* operations. The *search* operation, although also relevant, had to be omitted due to space restrictions. However it follows our previous work on the topic [11].

Algorithm 1: PUT Operation

Input: $DATA$

1 **begin**
2 $newIndexEntries \longleftarrow processAndIndex(DATA)$;
3 $HomEncrypt(newIndexEntries, PaillierKey, SearchKey)$;
4 $seed \longleftarrow random()$;
5 $keyGenerator \longleftarrow KeyGenerator(seed)$;
6 $TSS \longleftarrow generateThresholdSignatureShares(DATA)$;
7 $SSS \longleftarrow generateSecretSharingShares(seed)$;
8 $cloudObject \longleftarrow CloudObject()$;
9 **for** $i \leftarrow 1$ **to** $|C|$ **do**
10 $K_i \longleftarrow keyGenerator.next()$;
11 $DATA'_i \longleftarrow encrypt(DATA, K_i)$;
12 $replica_i \longleftarrow DATA'_i||TSS_i||SSS_i||TSS.PubKey$;
13 $RRef_i \longleftarrow SHA1(replica_i)$;
14 $cloudObject.Add(RRef_i, i)$;
15 **end**
16 $masterKey \longleftarrow keyGenerator.next()$;
17 $cloudObject' \longleftarrow encrypt(cloudObject, masterKey)$;
18 $masterRef \longleftarrow SHA1(cloudObject')$;
19 **for** $c_i \in C$ **do**
20 $c_i.PUT(RRef_i, replica_i)$;
21 $c_i.PUT(masterRef, cloudObject')$;
22 **end**
23 $storeReferenceIndex(masterRef, masterKey)$;
24 **end**

Algorithm 1 – *Algorithmic support for the put operation of a new email*

Sending a mail message. Algorithm 1 describes the procedure for storing a new mail message in the repository. When a message is sent trough the SMTP endpoint or via the external *put* operation in the Rest API, it is delivered to the Mailbox Manager (see Figure 1), which is charged of processing the mail message, extracting attachments, metadata and message contents. This information is then processed and indexed, resulting in the storage in TMS's indexing structures of the relevant metrics **(line 2)**. More details on how indexing is done can be found in [11]. To conclude this step, the new index entries are then encrypted with the homomorphic schemes discussed in section 3.1 **(3)**. The cryptographic keys required in this step are only used for encryption of the indexing structures and are also replicated securely through the storage clouds by means of the secret sharing mechanism. Additionally, these keys can be refreshed periodically through a key refreshment mechanism. Once the indexing is done, the Mailbox Manager request a data *put* internal operation to the layer below, with all the message data. This results in the required steps in order to replicate the mail message and distribute the resulting replica blocks through the available c clouds in a Byzantine quorum **(lines 4-22)**. Briefly, the algorithm generates a set of keys from a cryptographic seed **(10)** and encrypts all replicas with a different key **(11)**. The data is then attached to a share of the created threshold signature **(6),** a share of the seed used in the key generation process **(7)** and a copy of the threshold signature public key **(12)**. Each replica built this way is referenced by a hash digest **(13)** and this reference is stored in the new Cloud Object created **(14)** (see section 2.3 for more details). The Cloud Object is then encrypted with a master key **(17)** and it's replicated through the c clouds, along with each of the c replica blocks created before **(19-21)**. The algorithm concludes with the storage in the reference index of the Cloud Object's reference and respective master key **(23)**. This reference index is also stored encrypted and the respective key replicated through the secret sharing mechanism.

Algorithm 2: GET Operation

Input: $masterRef, masterKey$
Output: $DATA$

1 **begin**
2 $cloudObject' \longleftarrow c_x.GET(masterRef) : x \in C$;
3 $cloudObject \longleftarrow decrypt(cloudObject', masterKey)$;
4 **for** $i \leftarrow 1$ **to** K **do**
5 $replica_i \longleftarrow c_i.GET(RRef_i)$;
6 **if** $(SHA1(replica_i)! = RRef_i)$ **then**
// corrupted replica
// ignore replica
7 **else**
// continue
8 **end**
9 $TSS_i \longleftarrow replica_i.TSS$;
10 $SSS_i \longleftarrow replica_i.SSS$;
11 $DATA'_i \longleftarrow replica_i.DATA'$;
12 $TSS.PubK \longleftarrow replica_i.TSS.PubKey$;
13 **end**
14 $seed \longleftarrow recoverSSSecret(SSS)$;
15 $keyGenerator \longleftarrow KeyGenerator(seed)$;
16 **foreach** $DATA'_i$ **do**
17 $DATA_i \longleftarrow decrypt(DATA', keyGenerator.next())$;
18 $isValidData \longleftarrow checkTSScheme(DATA_i, TSS, TSS.PubK)$;
19 **if** $(isValidData)$ **then**
20 **return** $DATA_i$;
21 **else**
// continue
22 **end**
23 **end**
// unable to recover valid DATA
24 **end**

Algorithm 2 - *Algorithmic support for the get operation of an existing mail message, after retrieval of its Reference and Cryptographic key*

Receiving a mail message. When a message fetch is requested via the POP3 endpoint (or through the *get* operation in the Rest API), the request is forwarded to the Mailbox Manager. Once the Manager obtains the relevant data (master reference and cryptographic key for the Cloud Object of the fetched mail) it invokes a GET request on TMS's core. The core layer then proceeds as described in Algorithm 2. Briefly, the algorithm starts by recovering and decrypting the Cloud Object of the requested mail from one of the available clouds **(2-3)**. Any cloud can be chosen, as this object is replicated through all. Then, TMS retrieves all the replicas referred in the Cloud Object, validating their integrity **(5-8)** and extracting the stored signature and seed shares **(9-12)**. From the different seed shares the original seed is reconstructed **(14)** and the different replica blocks are decrypted **(17)**. To conclude, the replicas' integrity is validated through the

threshold signatures recovered **(18-22)**. A non-poisoned replica (if any was possible to recover) is then passed to the Mailbox Manager, which returns it to the client through the POP3 endpoint or Rest API.

4. PROTOTYPING AND EVALUATION

A prototype of TMS was developed in Java, including connectors for Amazon S3, Nirvanix Cloud Storage, Rackspace Cloud Files and Google Cloud Storage (the ICSPs chosen for integration with the prototype and for experimental evaluation of TMS). Versions of the secret sharing algorithms, threshold signatures and homomorphic encryption schemes (discussed in section 3) were also implemented in Java and resourcing to its standard libraries. The implemented prototype was then evaluated in a limited scenario, with a focus on performance metrics. The testing environment was set by running TMS locally in a quad core processor Intel Core i7-3630QM at 2.40GHz and a JVM maximum heap space of 512MB. For cloud access, experimental evaluation was carried over a network of 100 Mbits of maximum upload/download. The evaluation was carried in Lisbon and a set of heterogenic cloud providers was chosen (Amazon: Ireland; Nirvanix: California US; Rackspace: Dallas US (Rackspace's data-center in London wasn't available for the Cloud Files service); Google: EU/US data-centers).

In this section we present a brief evaluation of TMS through a set of end user performance tests. These tests aimed in the extraction of performance metrics from an email client, sending and receiving messages to and from TMS's prototype. Each message received by the prototype was redundantly stored in the four different storage clouds discussed before, as defined in TMS data model (section 2.3). The tests were divided in the following datasets: sending and receiving 1.000 and 10.000 messages. As a baseline for comparison of performance, we used the Gmail service. It should be noted that, due to limitations in this service, it was impossible to conduct tests in the baseline with more than 1.000 messages, meaning that in the biggest dataset we were not able to experimentally compare TMS with the baseline. However, the performance of the baseline would be expected to grow linearly with the dataset increase, as occurred with TMS's results. The datasets were taken from the Enron database of online messages [13], containing real emails of up to 200Kbytes each.

Figures 3 and 4 shows us the results of sending and receiving mail messages through TMS and compared with the baseline Gmail service. The evaluation of TMS was divided in 3 main processing parts: endpoint metrics, which consist in the latency between the client's MUA and TMS, through the SMTP/POP3 interfaces; core execution metrics, which include message processing time, indexing and cryptographic operations; and cloud performance times, which measure the aggregated cost of replicating data through the 4 clouds in parallel, as defined in our Byzantine model. It should be noted that the endpoint and core performances in message receiving include cloud operations due to the synchronous nature of message retrieval operations (which does not occur in message sending, as TMS can return success to the email client in an asynchronous way). Additionally, two versions of TMS were prototyped and evaluated, for performance comparison: the full version of TMS as described so far (TMS-TS), and a more simplistic version where the threshold signatures were replaced by conventional MAC signatures (TMS-MAC).

Comparing the three main processes of TMS, both in terms of send and receive operations, we can see that the heaviest process corresponds to cloud operations. This means that cloud latencies dictate the overall performance of TMS and that TMS algorithms themselves are very fast when compared to these cloud latencies. In terms of comparison with the baseline Gmail service, the baseline accomplishes worse times when compared to TMS in the message sending operation, in terms of the client's point of view (endpoint metrics). These results can be explained by a possibly synchronous message delivery mechanism held by the Google service (contrary to the asynchronous service of TMS).

On the other hand, in terms of receive operations the performance of TMS Endpoint is aggravated, as this operation cannot be performed in an asynchronous way due to its nature. In this operation, 99% of the latency perceived by the user comes from the cloud operations needed to deliver a message (in Figure 4 these percentage differences have been filtered). This could dramatically be reduced by using better low latency/high bandwidth cloud storage services, local caching techniques or using the cloud just as a redundant support for storing email data. The overall times are 10 to 15 times worse than the ones involved in Gmail service. In these receive operations, endpoint times become negligible considering the time constraints imposed by synchronous necessities and the latency introduced by the clouds.

Comparing the performances of TMS-MAC and TMS-TS approaches, we notice that for message sending operations we get core processing times 3 to 35 worse by using threshold signature schemes (TMS-TS). In terms of message receiving times, the results are only around 1.5 times worse in TMS-TS, which can be considered acceptable having in consideration the advantages gained in an asynchronous Byzantine setting as modeled in section 2.

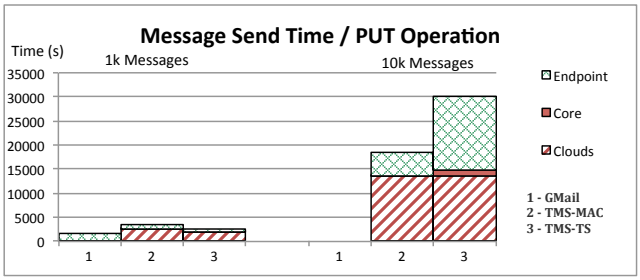

Figure 3 - *Message send times of GMAIL service, TMS-MAC and TMS-TS*

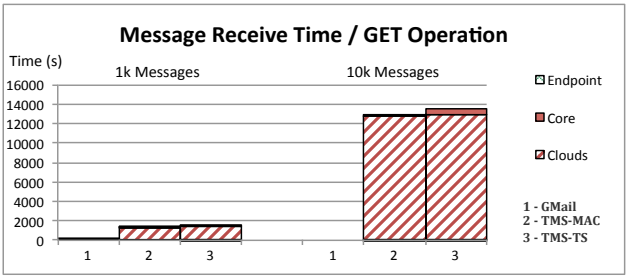

Figure 4 - *Message receiving times of GMAIL, TMS-MAC and TMS-TS*

5. RELATED WORK

The TMS architecture follows a cloud-of-clouds model in which diverse untrusted cloud storage services are used as a storage backend in a secure, reliable and dependable email repository solution. Data outsourcing systems are traditionally addressed by network file systems [14, 15]. Authentication and access-control services allow correct clients to mount locally file systems stored at the server, accessing remote files for transparent use. In these systems, the server is a trust computing base, supporting authentication functions and enforcing access control policies over the user's stored data. Cryptographic file systems [16, 17, 18] improve security guarantees, under the assumption that remote storage services are not necessarily trustable to provide

confidentiality, privacy, data-authentication or data-integrity properties to the clients. In cryptographic file systems all the data operations are done at the client side, where encryption/decryption takes place. Some cryptographic file systems [17, 18] also add file-sharing facilities, provided by means of an authenticated key distribution service. The TMS system is supported by remote untrusted clouds, organized in a cloud-of-clouds architecture, inspired by the relevant research work on cloud-security models and dependability solutions [3, 4]. As in [4], the TMS system adopts a cloud-of-clouds architecture providing operations for single-writer multi-reader read/write objects containing email messages. These objects are replicated on diverse untrusted storage clouds that can fail or may be attacked arbitrarily. The TMS system is however particularly addressed to build a middleware solution implementing a mail repository service, allowing the transparent integration of mail user agents implementing SMTP and POP3 protocols and allowing read/write/search operations of external applications over mailboxes and email messages. In the TMS system, security mechanisms implementing threshold-signatures [9] and secret-sharing techniques [6, 7, 8] are implemented as built-in middleware components, preserving guarantees of authenticity, confidentiality and integrity of mail messages, as private data.

Some data outsourcing models are based on the use of remote databases used as services (or DbaaS) [19]. These solutions allow clients to outsource structured databases maintained in cloud-supported databases, a model inspiring the current cloud-oriented DbaaS support model. These systems are focused in using remote SQL databases, not necessarily trusted. To support security and privacy guarantees, it is necessary to provide the necessary support for client execution of SQL encrypted queries over remote encrypted data. The use of homomorphic encryption schemes allows this solution. Some interesting approaches, like CryptDB [10] show that the support for SQL-based operations over private encrypted databases running in untrusted servers is possible, with an interesting balance between security and performance, requiring only partial homomorphic schemes and avoiding the overhead or the practical impossibility of fully or complete homomorphic encryption algorithms. TMS is mainly focused in exploring partial homomorphic encryption schemes to provide the relevant operations provided by email-storage systems and allowing ranking-oriented searching over private mailboxes maintained in multiple key-value stores, as offered by Internet Cloud-Storage Providers (ICSPs).

6. CONCLUSIONS

In this paper we addressed the design and implementation of TMS – an interoperable middleware architecture providing a trusted email repository service with security, privacy, availability and reliability guarantees, using a storage backend implemented by multiple untrusted cloud solutions in a cloud-of-clouds architecture. The solution offers external services as provided by conventional email repositories, supporting Mail User Agents or WebMail applications implementing SMTP or POP3 standard operations (over SSL or not). TMS adds security, privacy, availability and reliability guarantees, controlled by the user, and is designed to run as a local proxy in a client machine or as a trusted remote proxy as a service. The TMS implementation shows the feasibility of its design options. The evaluation demonstrates interesting and promising results for latency and performance, revealing that the impact introduced by the TMS middleware processing is modest and clearly compensates the additional dependability guarantees offered to the users.

7. REFERENCES

[1] Box Sentry. “Email Integrity: An Emerging Business Issue”. December 2009. White Paper. http://www.trustsphere.com/wp-content/uploads/2011/10/Gartner-Email-Integrity-Dec09.pdf

[2] I. Ion, N. Sachdeva, P. Kumaraguru, S. Capkun, Home is Safer than the Cloud: Privacy Concerns for Consumer Cloud Storage, Proc. of SOUPS 2011, Symposium on Usable Privacy and Security, Pittsburgh, 2011

[3] P. Verissimo, A. Bessani, M. Pasin. The TClouds architecture: Open and resilient cloud-of-clouds computing, IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W), June, 2012

[4] A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa. DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds. EuroSys’11, April 10–13, 2011, Salzburg, Austria

[5] A. J. Menezes, P. C. Oorschot and S. A. Vanstone, "Secret Sharing," in Handbook of Applied Cryptography, 1996, pp. 524-528

[6] A. Shamir, "How to Share a Secret," Communications of ACM, vol. 22, no. 11, 1979.

[7] K. Bozkurt and G. Selcuk, “Threshold Cryptography Based on Blakely Secret Sharing,” Information Sciences, 2008.

[8] K. Kaya, S. A. Aydın and Z. Tezcan, "Threshold Cryptography Based on Asmuth-Bloom Secret Sharing," 2007.

[9] V. Shoup, "Practical Threshold Signatures", EUROCRYPT'00, pp. 207-220, 2000.

[10] R. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. SOSP ’11, October 23–26, 2011, Portugal.

[11] B. Ferreira and H. Domingos. 2013. Searching Private Data in a Cloud Encrypted Domain. In Proceedings of the 10th International Conference in the RIAO series (OAIR 2013).

[12] P. Paillier. “Public - key cryptosystems based on composite degree residuosity classes”. In Proceedings of EUROCRYPT’99, Prague, Czech Republic, May 1999.

[13] Klimt, Bryan, and Yiming Yang. "Introducing the Enron Corpus." In CEAS. 2004.

[14] S. Shepler, B.Callaghan, D.Robinson, R.Thurlow, C.Beame, M. Eisler, and D. Noveck. NFS version 4 protocol. RFC 3530, April 2003.

[15] J.Howard. An Overview of the Andrew File System. Proceedings of ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2002.

[16] C. Wright, J. Dave and E. Zadok. "Cryptographic file systems performance: What you don’t know can hurt you." In SISW'03, pp. 47-47. IEEE, 2003.

[17] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing remote untrusted storage. Proceedings of Network and Distributed Systems Security (NDSS) Symposium, 2003.

[18] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. Proceedings of USENIX FAST’03, 2003.

[19] H.Hacigumus, B.Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. ACM SIGMOD Conference on Management of Data, Jun, 2002.