A Secure Email Repository Service using Public Untrusted Storage Clouds

João Rodrigues, Bernardo Ferreira, Henrique Domingos

Departamento de Informática, FCT/UNL
CITI Centro de Informática e Tecnologias da Informação
jm.rodrigues@campus.fct.unl.pt, bernardof@acm.org, hj@fct.unl.pt

Abstract. In this paper we present the TMS or Trusted Mail System, a dependable Email repository service that explores multiple untrusted cloud storage repositories 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.

Keywords: Untrusted Cloud Storage; Threshold Signatures; Secret Sharing; Homomorphic Encryption; Email Security and Reliability; Ranked Searching.

1 Introduction

Internet cloud storage providers (ICSPs) are currently used as data outsourcing services, continuously growing in the market. ICSP solutions, including Amazon S3, Dropbox, Yahoo Briefcase, Google Drive, Windows Azure, Rackspace, Nirvanix or CloudPT, among others, provide data management tasks, highly-available elastic storage and ubiquitous data access. These solutions are offered with appellative pay-per-use cost models, targeted for different requirements of storage sizing and read/write operations volumes.

In general, ICSP solutions are provided without service level agreement (SLA) conditions controlled by the end user. Security, availability and reliability guarantees under user auditing criteria, regulation and compliance verification is not possible. With so many different cloud-deployment solutions available today, no available security or reliability controls can cover all the circumstances and requirements. As such, the decision to move private data to the cloud is only possible in a risk-based approach. This requires a specific framework guide to evaluate, carefully, the dependability concerns of cloud-move decisions in each case [1, 2].

A particular context of data outsourcing on ICSPs is the case of Email services for individual, corporative or institutional use. These solutions tend to be used as outsourced Email repositories for Internet ubiquitous access, using Webmail or conventional Mail User Agents or MUA applications rutting in different user devices. Most companies consider email to be a mission critical application. 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. However, it is usual to observe a clear contradictory approach in the way how cloud-based email outsourcing services as commonly adopted: in one hand, cloud storage services provide no dependability properties under the control of end users; in the other hand, many studies ranked security and privacy to be major areas of concern and obstacles to adopt cloud solutions [3].

The use of multiple storage clouds to materialize transparent cloud-of-clouds data repository architectures is a challenging direction. This approach allows leveraging and improving reliability, availability and security conditions in the design of dependable cloud-based storage services. These solutions benefit from the resilience conditions established by the diversity of multiple clouds and 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 [4]. It is also an interesting design option in addressing intrusion-tolerance, leveraging from hardware/software heterogeneity and independent failures/attacks in each individual cloud[5]. Dependable cloud-of-clouds architectures for scalable data-storage services, combining privacy, integrity, availability and reliability properties, can be used for different scenarios for data outsourcing, without outsourcing data control.

Inspired from the relevant work on dependability services in the design of cloud-of-clouds data storage architectures [4, 5], this paper addresses the design and implementation of TMS (Trust Mail System), an email repository service, using a storage backend built as a cloud of internet-based storage clouds as repository components as offered by current ICSPs. TMS architecture is designed and implemented as an interoperable middleware solution, outsourcing mailboxes and mail messages, mapped as private objects distributed and replicated in diverse untrusted clouds. The goal is to offer, on top of TMS, similar services as offered by usual outsourced (and untrusted) email repository services and platforms, supporting data repository services for Mail User Agents or Webmail applications. TMS adds security, privacy, availability and reliability guarantees, controlled by the user. The solution is designed to run as a local proxy (in a client machine) or as a trusted remote proxy (used as a trusted service). The system provides SMTP, POP3 and REST standard services and offers an API that translates data related write/read/search operations on mailboxes and contents, to the equivalent operations in the backend storage supported in a cloud-of-clouds architecture.

2 TMS System Model and Architecture

2.1 System model

As introduced before, the system model follows a design oriented to a cloud-of-storage-clouds architecture. This model addresses a middleware solution between Mail User Agent applications (MUAs) or Webmail 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 IETF RFCs 2595 and 3207). The TMS service is also available via a Web REST-based API, providing read/write/search operations over objects corresponding to electronic 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 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, respectively.

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, integrating 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 we admit that a cloud write operation only fails with a fail stop model. The interaction between TMS and the backend clouds supports a replication process, in which the TMS processed mail messages are replicated in a number of storage clouds. As we consider that backend clouds are not trusted computing bases (see the adversary model in the next section), all replicas are stored encrypted.

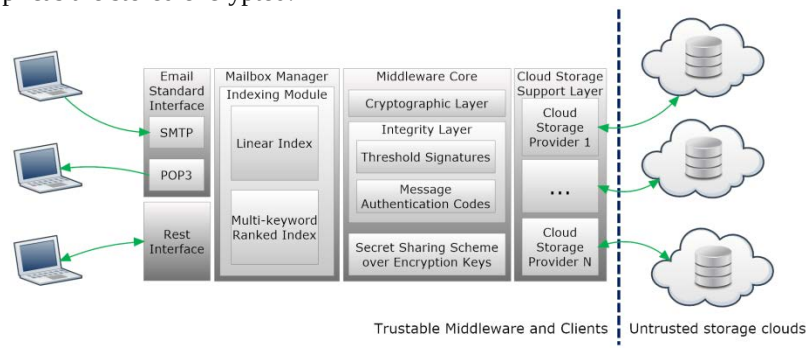

Figure 1 - TMS Architecture and System components

Figure 1 presents the TMS architecture and system components. Additionally to the referred APIs and backend Cloud Connectors, different components in the mid-

dleware core implement the indexing mechanisms and cryptographic algorithms required to support the TMS functionalities. These mechanisms and their justification are discussed in more detail in section 3.

2.2 Security Considerations 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 internal and/or external attacks or intrusions. On the other hand, the local servers where the TMS system is deployed are controlled by the clients and are considered a secure computing base. We assume that outsider and insider attacks can take place against each storage cloud adopted in the TMS storage backend, compromising data stored in such clouds. We consider insider attacks, resulting from malicious operations from employees or system administrators, as adversary hypothesis. We also consider outsider attacks, resulting from malicious actions executed by Internet hackers acting as intruders breaking the storage provider's system and stealing or corrupting the user's data with a Byzantine attack setting. The only restriction is that the attacks against each cloud follow an independent model (e.g., the attacks are not correlated in any way under the correlation control of the same attacker). To support reliability and intrusion tolerance the TMS system will adopt a set of $3f+1$ untrusted storage clouds, for a resilience support against f faulty (or attacked) clouds.

2.3 Back-End Storage Services 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 the data model. As seen in the left side 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 of three indexes:

- The reference index: co-relates message *Ids* with tokens composed of: 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;
- A multi-keyword ranked homomorphic search index: allows search operations over encrypted email message contents while preserving privacy, i.e., the disclosure of the locally stored index by itself doesn't reveal any message data. A search in this index returns a set of unique message identifiers translated to in-cloud references through the reference index;
- A Boolean index: allows fast searching over email header fields of standard Email message formats (defined in RFC 5322), including recipients, sender or subject. This index, as the previous one, returns a set of unique message identifiers that are translated to in-cloud references through the reference index;

In the storage clouds (right part of Figure 2), two data structures are required in the designed data model:

- A Cloud Object, which represents an email message (one Cloud Object per message) and references 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;
- 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 a set of encryption keys; a share of the threshold signature; and a public key for the threshold signature verification process.

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

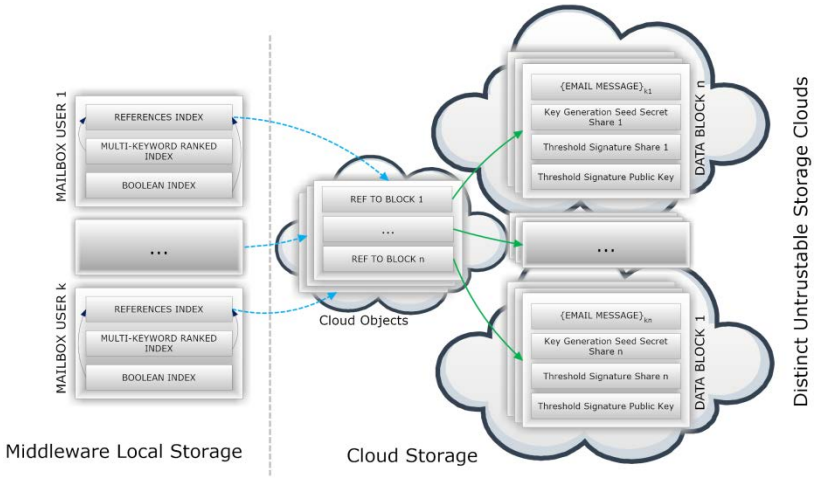

Figure 2 – TMS Data Model

3 System Components

In this section we describe the mechanisms and algorithms implemented in the TMS middleware services. These mechanisms compose the different components in the TMS middleware core, as described in section 2.1 and represented in Figure 1.

3.1 Secret Sharing and Threshold Signatures Schemes

In *threshold(t,n)* based secret sharing schemes [6] information is splitted in multiple parts or shares in a way that only with a minimum of *t* parts out of *n* is possible to recover the secret, safeguarding cryptographic keys from loss without the need of creating backup copies. Our solution approach is leveraged by these schemes exploring them in a Byzantine model where *f* shares out of *3f+1* can be corrupted, in which

f=n-t, this is, this scheme safeguards the keys from loss or disclosure attacks. In order to securely explore secret share scheme, given the knowledge of *t-1* or fewer shares should be impossible to get any clue about the shared secret. The developed solution was primarily tested using each one of the three different secret sharing schemes. The Shamir Secret Sharing Scheme [7] based in Lagrange Interpolation; Blakley Secret Sharing Scheme [8] based on hyperplanes intersection; and Asmuth-Bloom Secret [9] based on modular arithmetic. The TMS solution was evaluated using Blakley Scheme due to its stability, furthermore, in our previous experimental evaluation and comparison between different secret sharing schemes showed they possess similar performances.

A threshold signature scheme [10] is an asymmetric signature process that allows the creation of a set of signature shares instead of a single signature, as defined in RSA based signatures. This scheme, so-called a threshold(t,n) signature scheme, based on RSA and Lagrange interpolation, requires a public key and at least *t* out of *n* signature shares to be able to verify the signature. Once a set of private key shares and a public key is generated, each private key share is then used to sign the same data block. Then a verifier is added to each share individually, allowing the detection of corrupt shares and avoiding poisoning in the signature verifying process, giving strong authenticity guarantees. As in secret sharing schemes our solution approach is leveraged by these schemes exploring them in a Byzantine model where *f signature* shares out of *3f+1*, in which *f=n-t*, can be corrupted ensuring the existence of an integrity and authenticity verification process.

3.2 **Homomorphic Encryption Schemes**

Homomorphic encryption schemes are encryption algorithms that allow certain operations to be executed over the encrypted data, without having to decrypt it and while preserving its privacy [11]. This is apparently a contradictory aspect regarding the security properties of the cryptographic algorithms themselves, meaning that a good homomorphic encryption scheme must be able to combine its homomorphic properties while preserving required security characteristics [12].

In the context of our work, the need to 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 implementation and employment of a partially homomorphic scheme. This encryption scheme (Paillier [13]), based on modular arithmetic properties, supports additions over the ciphertext and has a security level of IND-CPA. We point the reader to [12] and [13] for more details on the Paillier scheme. Through the application of this scheme it is possible to encrypt relevance scores stored in the TMS indexes and preserve the ability to perform searches over these (explained in more detail in the next section).

3.3 **Indexing and Searching**

In the system architecture, a main component is the searching and indexing module. This component allows the TMS system to perform Boolean as well as ranked

searches over the email repository while protecting their privacy. This is achieved by using a combination of information retrieval techniques and homomorphic encryption schemes.

In order to allow searching email messages from the repository, the system maintains two indexing structures. The first index (Boolean index) stores email metadata information, allowing clients to perform fast Boolean searches. An example of search allowed by this index would be "all email messages sent by Alice". The second indexing structure (ranked index) is directed to email content and subjects, and allows ranked multi-keyword searching. This is accomplished through the use of inverted list indexes, posting lists and probabilistic scoring [12].

The indexing structures are updated each time a new email message is sent (or injected) by TMS clients and before their secure and reliable storage in the Clouds. In order to minimize exposure of the indexed metrics in case of an attack to the TMS infrastructure, the indexes are kept encrypted at all times. In particular, the ranked index is encrypted with the Paillier scheme, previously introduced in the section 3.2). This allows searches to be carried over the index without having to decrypt the ranked metrics and while preserving privacy guarantees.

3.4 System Processing and Algorithms to store and retrieve Email messages

The processing model of email messages in the TMS middleware is based in two processing flows, expressing the most basic mailbox operations used as proof of concept, represented in Algorithms 1 (store or put messages) and 2 (retrieve or get messages), also described in the next paragraphs.

Algorithm 1: PUT Operation

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

begin
  $seed \longleftarrow random()$;
  $keyGenerator \longleftarrow KeyGenerator(seed)$;
  $TSS \longleftarrow generateTSignatureShares(DATA)$;
  $SSS \longleftarrow generateSecretSharingShares(seed)$;
  $cloudObject \longleftarrow CloudObject()$;
  for $i \leftarrow 1$ **to** $|C|$ **do**
    $K_i \longleftarrow keyGenerator.next()$;
    $DATA'_i \longleftarrow encrypt(DATA, K_i)$;
    $replica_i \longleftarrow DATA'_i || TSS_i || SSS_i || TSS.PubKey$;
    $RRef_i \longleftarrow SHA1(replica_i)$;
    $cloudObject.Add(RRef_i, i)$;
  end
  $masterKey \longleftarrow keyGenerator.next()$;
  $cloudObject' \longleftarrow encrypt(cloudObject, masterKey)$;
  $masterRef \longleftarrow SHA1(cloudObject')$;
  for $c_i \in C$ **do**
    $c_i.PUT(RRef_i, replica_i)$;
    $c_i.PUT(masterRef, cloudObject')$;
  end
end

Algorithm 2: GET Operation

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

begin
  $cloudObject' \longleftarrow c_x.GET(masterRef) : x \in C$;
  $cloudObject \longleftarrow decrypt(cloudObject', masterKey)$;
  for $i \leftarrow 1$ **to** K **do**
    $replica_i \longleftarrow c_i.GET(RRef_i)$;
    if $(SHA1(replica_i)! = RRef_i)$ **then**
      // corrupted replica
      // ignore replica
    else
      // continue
    end
    $TSS_i \longleftarrow replica_i.TSS$;
    $SSS_i \longleftarrow replica_i.SSS$;
    $DATA'_i \longleftarrow replica_i.DATA'$;
    $TSS.PubK \longleftarrow replica_i.TSS.PubKey$;
  end
  $seed \longleftarrow recoverSSSecret(SSS)$;
  $keyGenerator \longleftarrow KeyGenerator(seed)$;
  foreach $DATA'_i$ **do**
    $DATA_i \longleftarrow decrypt(DATA', keyGenerator.next())$;
    $isValidData \longleftarrow checkTSScheme(DATA_i, TSS, TSS.PubK)$;
    if $(isValidData)$ **then**
      // return valid DATA
      $DATA = DATA_i$;
    else
      // continue
    end
  end
  // unable to recover valid DATA
end

Sending a message. When a message is sent (or injected in the TMS) through the SMTP or REST endpoints, it is delivered to the mailbox manager that is charged of extracting all the words from message body and attachments inserting them into the search index. Once the indexing is done the mailbox manager perform a PUT operation request to the layer below, with all the message data. Then the core proceeds as described in Algorithm 1, returning a master key and reference that is inserted into reference index within mailbox layer and stored locally on disk for persistency purposes. Each of encrypted email replicas along integrity proofs are then stored in the different backend clouds.

Receiving a message. When a message fetch is requested via the POP3 or REST endpoints the request is forwarded to the mailbox manager, detaining all the needed data to recover the message from the storage clouds. Once the mailbox manager obtains the master keys and references from the index associated with a particular user mailbox, the mailbox manager invokes a GET operation over core layer, providing master references and keys needed to retrieve the message. The core layer then proceeds as described in Algorithm 2, requesting the data from the multiple clouds used and applying the operations used in Algorithm 1 in reverse order and returning the plain email to the mailbox manager which returns it to the POP3 and REST endpoints.

4 TMS 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. Versions of the secret sharing algorithms, threshold signatures and homomorphic encryption schemes (discussed in section 3) were also implemented.

The implemented prototype was used for experimental evaluation, aiming at performance metrics. The testing environment was set by running the middleware system 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 the Eduroam wireless network, which best illustrates a real use case scenario.

In this section we present a brief evaluation of TMS service through a set of end user performance tests. These tests basically consist in the extraction of performance metrics from an email client, sending and receiving messages to and from the presented system. Each of messages received by middleware is redundantly stored in 4 different storage clouds (Google Cloud Storage, Amazon S3, Nirvanix Cloud Storage and Rackspace Cloudfiles) as defined in data model section. The tests were divided in the levels: first sending and receiving 10 messages, 100 messages, 1000 messages and finally 10000 messages. Due to limitations in Gmail service it was impossible to conduct tests with more than 1000 messages. The used subset of email messages taken from an online dataset[1] contains messages from 1Byte to 200Kbytes with plain text content. The conducted tests will allow us to reason about service acceptance compared with today wide spread well known email services.

[1] Available at: http://www.cs.cmu.edu/~enron/ [Accessed on 24 June 2013]

Table 1 shows us three types of metrics: the endpoint metrics consisting in the latency observed by end user (like in Gmail service); the core execution metrics include message processing time, indexing and cryptographic operations. The endpoint and core performance in message receiving case include over-cloud operations due to the synchronous needs, i.e., unlike in message send operations, in message retrieval operations the system needs to retrieve and process message from the clouds before returning it to client. Still, in message send case, the endpoint and core performance do not include over-cloud operation times since once the message is delivered to the system the endpoint return success to email client. The cloud performance times represent the operation execution latency over the slowest cloud (due to parallel cloud requests).

# Msgs	Test Type	GMAIL		TMS-MAC		TMS-TS	
		Send	Receive	Send	Receive	Send	Receive
10	Endpoint	18,108	1,327	1,44	13,5	2,55	18,97
	Core	-	-	0,456	13,455	1,606	18,932
	Clouds	-	-	20,2696	13,31	23,7148	18,063
100	Endpoint	187,521	12,989	10,48	131,09	23,12	133,48
	Core	-	-	1,106	130,992	11,727	133,422
	Clouds	-	-	467,188	130,288	235,289	126,702
1 000	Endpoint	1821,059	127,396	663,92	1240,33	734,76	1502,42
	Core	-	-	6,578	1240,258	76,344	1502,433
	Clouds	-	-	2694,173	1235,933	1845,449	1437,302
10 000	Endpoint	-	-	4812,64	12709,34	15302,61	13439,83
	Core	-	-	53,934	12726,06	1182,538	13506,49
	Clouds	-	-	13605,91	12685,67	13741,3	12848

Table 1 - Performance (measured in seconds) and comparison of GMAIL email service and the impact of TMS middleware when using the TMS-MAC and Threshold Signatures variants

Through a brief observation of the Table 1 we verify that Gmail service accomplishes worse times in message sending operations on the endpoint point of view. This fact can be backed by a synchronous message delivery mechanism held by Google service. Although we cannot take this fact as a performance advantage, today email services include message receipts which tell users the delivery state of their messages. In the presence of a synchronous send message operations it would be expected worse times as seen in Figure 3 where the overall time is 1.1 to 2.5 times worse, mainly due to over-cloud operations. Nonetheless the presented times can be considered acceptable, given that 1 to 5 seconds to send an email is acceptable.

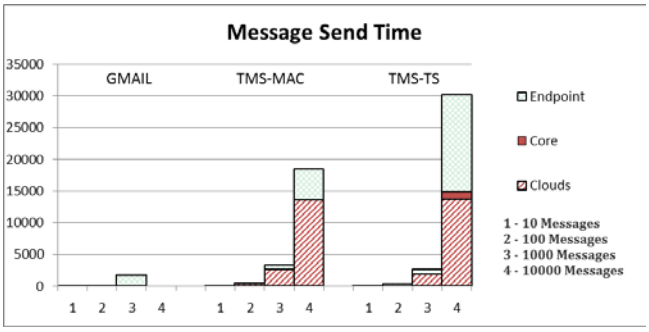

Figure 3 - Message send times to GMAIL service, TMS-MAC and TMS-TS

On the other side we verify that email receiving times are worse in our system. The message deliver time is basically determined by over-cloud operations (Figure 4) taking about 99% of all time needed to deliver a message to end client. This time could dramatically reduce by using better low latency cloud storage services, caching techniques or using the cloud just has a redundant support for storing email data. The overall times are 10 to 15 times worse than the ones involved in Gmail service.

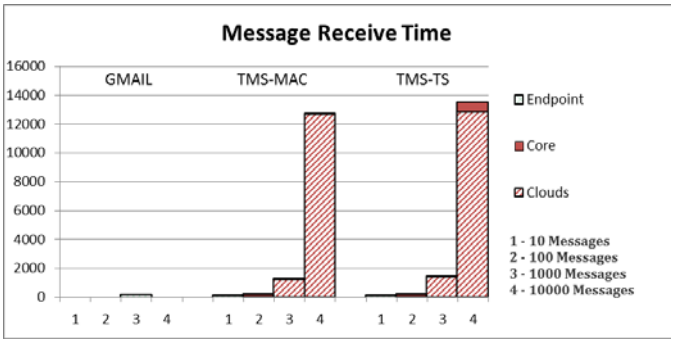

Figure 4 - Message receiving times to GMAIL service, TMS-MAC and TMS-TS

Comparing the use of TMS-MAC approach where instead of Threshold Signatures schemes Message Authentication Codes are used as data integrity proofs, versus TMS-TS we notice that for message sending operations we get core processing times 3 to 35 worse in TMS-TS then in TMS-MAC. Despite this the message receiving times are just up to 1.5 worse in TMS-TS which can be considered acceptable.

Unlike in synchronous receiving times, the considerable endpoint performance times in message sending operations exist due to an artificially added constraint so the TMS service would not drown under a large number of requests.

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. 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 [14, 15, 16] improve security guarantees, under the assumption that remote storage services are not necessarily trustable to provide confidentiality, privacy and data authentication or 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 [15, 16] 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 [4, 5]. As in [5], 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 are 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 or POP3 protocols and allowing read/write/search operations of external applications over mailboxes and email messages. In TMS system, security mechanisms implementing threshold signatures [10] and secret sharing techniques [7, 8, 9] 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) [17]. 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 [11] 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 clouds solutions in a cloud-of-clouds architecture. The solution offers external services as provided by conventional email repositories, supporting MUAs 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 not significant and clearly compensates the additional dependability guarantees offered to the users.

References

1. CSA – Cloud Security Alliance, Security Guidance for Critical Areas of Focus in Cloud Computing (v3.0), 2011 (available in https://cloudsecurityalliance.org)
2. Richard Chow, P Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, *Controlling data in the cloud: outsourcing computation withpout outsourcoing control*, CCSW 2009, Proc. of the 2009 ACM Workshop on Cloud Computing Security. Chicago IL, USA, November 2009
3. Iulia 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
4. Paulo Verissimo, Alysson Bessani, Marcelo 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*
5. 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
6. A. J. Menezes, P. C. Oorschot and S. A. Vanstone, "*Secret Sharing,*" in *Handbook of Applied Cryptography*, 1996, pp. 524-528
7. A. Shamir, "How to Share a Secret," Communications of ACM, vol. 22, no. 11, 1979.
8. K. Bozkurt and G. Selcuk, "Threshold Cryptography Based on Blakely Secret Sharing," Information Sciences, no. x, 2008.
9. K. Kaya, S. A. Aydın and Z. Tezcan, "Threshold Cryptography Based on Asmuth-Bloom Secret Sharing," 2007.
10. V. Shoup, "Practical Threshold Signatures", EUROCRYPT'00, pp. 207-220, 2000.
11. R. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. SOSP '11, October 23–26, 2011, Cascais,Portugal.
12. Bernardo Ferreira and Henrique Domingos. 2013. Searching Private Data in a Cloud Encrypted Domain. In Proceedings of the 10th International Conference in the RIAO series (OAIR 2013).
13. P. Paillier. Public - key cryptosystems based on composite degree residuosity classes. In Proceedings of the 18th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Prague, Czech Republic, May 1999.
14. C. Wright, J. Dave and E. Zadok. "Cryptographic file systems performance: What you don't know can hurt you." In SISW'03. Proceedings of the Second IEEE International, pp. 47-47. IEEE, 2003.
15. E.Goh, H.Shacham, N.Modadugu, and D.Boneh. SiRiUS: Securing remote untrusted storage. Proceedings of Network and Distributed Systems Security (NDSS) Symposium, 2003.
16. M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. Proceedings of 2nd USENIX Conference on File and Storage Technologies (FAST), 2003.
17. 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.