IDIS (2009) 2:99–114
DOI 10.1007/s12394-009-0033-3

Trusting third-party storage providers for holding personal information. A context-based approach to protect identity-related data in untrusted domains

Giulio Galiero · Gabriele Giammatteo

Received: 12 January 2009 / Accepted: 28 September 2009 / Published online: 14 November 2009

Abstract The never ending growth of digital information and the availability of low-cost storage facilities and networks capacity is leading users towards moving their data to remote storage resources. Since users' data often holds identity-related information, several privacy issues arise when data can be stored in untrusted domains. In addition digital identity management is becoming extremely complicated due to the identity replicas proliferation necessary to get authentication in different domains. GMail and Amazon Web Services, for instance, are two examples of online services adopted by million of users throughout the world which hold huge amounts of sensitive users data. State-of-the-art encryption tools for large-scale distributed infrastructures allow users to encrypt content locally before storing it on a remote untrusted repository. This approach can experience performance drawbacks, when very large data-sets must be encrypted/decrypted on a single machine. The proposed approach extends the existing solutions by providing two additional features: (1) the encryption can also be delegated to a pool of remote trusted computing resources, and (2) the definition of the encryption context which drives the tool to select the best strategy to process the data. The performance benchmarks are based on the results of tests carried out both on a local workstation and on the Grid INFN Laboratory for Dissemination Activities (GILDA) testbed.

Keywords Distributed computing · Distributed storage · Secure data storage · Parallelized encryption · Grid computing · Sensitive data · Digital identity

G. Galiero (✉) · G. Giammatteo
Engineering Ingegneria Informatica S.p.A., Rome 00185, Italy
e-mail: giulio.galiero@eng.it

G. Giammatteo
e-mail: gabriele.giammatteo@eng.it

Introduction

The amount of digital data produced every year is growing exponentially. From the end-users' perspective, the digital world is a an unlimited virtual drawer where their entire life can be stored: emails and attachments, documents and spreadsheets, holiday pictures, music and videos. The more data are produced, the more the need for storage space increases. As a consequence, digital data preservation is becoming a thorny concern: users do not want to deal with managing such an amount of data, even though it is their private data. Thus users, rather than relying on a do-it-yourself data management, prefer to trust professional IT providers which offer support to outsource user data on remote servers (e.g. Google—http://www.google.com, Yahoo—http://www.yahoo.com, Facebook—http://www.facebook.com, Amazon—http://aws.amazon.com/, etc.). Online storage scenarios can also include the need for the end-users to transfer and store very large data set (i.e. magnitude of GiBs): hard drives backup by private users, phone call logs by a telco company, patients records and medical images (X-Rays, radiotherapy treatments) by healthcare institutions.

As people rely on banks to save their money rather than hiding it at home, users are now relying on third-parties to store their private data rather than using their laptops as data repositories. The main difference between money and data is that the latter brings along huge privacy issues related to the sensitive content they hold, which probably makes data much more valuable than money. However, users, no matter the privacy issues, are strongly attracted by the plethora of online services for two main reasons: (1) the anywhere, anytime, anyhow data access pattern is close at hand, (2) it is - nearly always - free. The result is that private sensitive data are drifting from local hard disks to remote storage sites. This trend is evidenced by the recent spawning of online storage services available on the web. The more online services are used, the more sensitive data are distributed throughout several storage providers. The Amazon S3 (Simple Storage Service), for instance, is the data storage service included in the Amazon Web Services (Murty 2008; Amazon—http://aws.amazon.com/) suite, which has proved to be very popular among the end-users. However, the data stored on the Amazon servers is not encrypted (http://s3.amazonaws.com/aws_blog/AWS_Security_Whitepaper_2008_09.pdf): the need for privacy is proved by the current availability of third-party tools (Dropbox—http://www.getdropbox.com/, Spideroak—http://www.spideroak.com/) that wrap the Amazon S3 functionalities and introduce additional encryption features. Unfortunately, as more services are available, the user digital identity is replicated, since new user accounts are created for each different service.

Several attempts have been put forward to streamline digital identity management in distributed environments (Kreizman et al. 2007; Cameron 2005). The basic idea behind these approaches is to extract identity-related information into an independent layer where users' profiles can be held. Thus, since identity is freed from the applications, the applications themselves can verify the identity of the users by talking to this new layer, where identity authorities are put in place. Several user-centric technologies have been delveloped to support such new scenarios (OpenId—http://openid.net/; Recordon and Reed 2006; Information Cards—http://informationcard.net/; Shibboleth—http://shibboleth.internet2.edu/). The main advantage of having only one digital identity reveals to be a drawback with regards to held identity-related data online. If each user is associated to a single digital id, then users' online activities can be entirely tracked back. Even though efforts have been put into trying to achieve users digital anonymity or pseudonymity (Rodrigues 2007) currently there is still high risk of identity correlation (Schneier—http://www.schneier.com/essay-200.html/) as well as identity theft. Other risks involve potential backdoor communications between third-parties, who can build up knowledge about their users by exchanging identity-related information between each other. OAuth (http://oauth.net/), for instance, is a solution aiming at reducing or, if possible, preventing this scenario: the main idea behind OAuth is to define a user-centric protocol which states the interaction between third parties and users in order to allow the latter to issue consent/dissent assertions on the behavior of the former.

The common approach to protect user information is to encrypt data before they are uploaded to an untrusted domain, as suggested, for instance, in the Amazon's Security Whitepaper (http://s3.amazonaws.com/aws_blog/AWS_Security_Whitepaper_2008_09.pdf). Our contribution is based on this assumption and furthermore aims at taking into account the encryption context in order to dynamically adapt the encryption operations.

The rest of this paper is organized as follows: We first discuss related work in section "Related work" and then analyze potential challenges and motivation toward secure storage in distributed environments in section "Challenges and motivation". In section "Design and implementation", we present our design and implementation in detail. We also carry out performance evaluation in section "Performance evaluation" to show efficiency comparison. Finally, we conclude the paper in section "Conclusions and future work".

Related work

Several tools have been developed to protect sensitive data in distributed environments. In this section three tools for grid distributed environments are presented. Each implementation is evaluated against the following features: (a) the support for local and/or remote encryption, (b) the pool of encryption algorithms supported, (c) the way the encryption keys are managed, (d) the support for Shamir Secret Sharing (1979). Further a proprietary data storage service is presented.

S3 (Scardaci and Scuderi 2007)—Secure Storage System—is one of the encryption services provided by the gLite (http://glite.web.cern.ch/glite/) middleware. S3 offers a security layer of abstraction to interact with files on the Grid, which extends both the gLite-utils and the GFAL (http://wiki.egee-see.

org/index.php/Data_Access_with_GFAL) (POSIX like) APIs. User applications can leverage the S3 APIs to encrypt/decrypt data locally (on a single node of the trusted domain) storing/retrieving data to/from remote storage nodes. The encryption process is always performed via the AES-256 standard algorithm. Key management is performed through the Keystore service, which enables sharing of encrypted data among the members of a VO through Access Control Lists (ACL). Shamir Secret Sharing support is not yet implemented, but has been included in the development roadmap. A similar service, named GS3 (https://grid.ct.infn.it/twiki/bin/view/Main/GridSecureStorageSystem)—Grid Secure Storage System—, has been developed by Consorzio Cometa (http://www.consorzio-cometa.it/). The most remarkable difference between two services is that GS3 doesn't support ACLs. Key Management is based on PKI certificates, instead.

Hydra (Montagnat et al. 2006; https://twiki.cern.ch/twiki//bin/view/EGEE/DMEDS) is a keystore server firstly developed for the AGIR (http://www.aci-agir.org/) project to protect medical digital images of patients, and then integrated into the gLite middleware. Data encryption is performed via the AES standard algorithm of the OpenSSL version shipped with the Globus Toolkit. Data are encrypted/decrypted locally, without any support for data chunks encryption/decryption. Key management relies on a set of keystore servers, which are the building blocks of the Hydra service ACLs, thus allowing principals to share encrypted files among a group of users. In addition, to increase security, Hydra supports also Shamir Secret Sharing (to split keys into a set of key chunks stored in different Hydra servers).

Perroquet (Blanchet et al. 2006) is a tool for grid environments developed by the EGEE (www.eu-egee.org) project to secure biological data. Perroquet is built on top of Parrot (http://www.cse.nd.edu/~ccl/software/parrot/) and EncFile (Blanchet et al. 2006) to provide an encryption client which allows "any applications to transparently read and write encrypted remote files as if they were local and plain-text". AES-256 is the algorithm used to encrypt data. Data chunks encryption is not supported. The Perroquet architecture relies on a set of key servers to store the encryption keys. Actually, since Perroquet supports Shamir Secret Sharing, the key servers hold only chunks of keys, not the key itself.

For each encryption tool reviewed in this section, a summary of the main features is provided in Table 1.

Table 1 State of the art

	Enc. location	Algorithms	Key management	Shamir Secret Sharing
S3	Locally	AES-256	Key server	Included in the roadmap
GS3	Locally	AES	PKI to encrypt user's enc keys	No
Hydra	Locally	AES	Key server	Yes
Perroquet	Locally	AES-256	Key server	Yes

Summary of reviewed tools

Challenges and motivation

Taking into account the encryption context

In a flexible secure storage service, the encryption strategy should depend on the encryption context. We define the encryption context as the set of values related to the following parameters: the input data size, the security requirements and the encryption speed requirements. However, state of the art tools provide users with a basic "no-choice" encryption strategy. In other words such tools offer only one encryption algorithm (i.e. AES, the standard), disregarding the encryption context.

In order to achieve better accuracy and efficiency, the encryption context should drive the choice of the best strategy; that is which encryption algorithm and key size should be used and whether the encryption should be either executed locally or parallelized on remote machines. Thus, a pool of encryption algorithms should be available to cover a wide range of scenarios. Once the encryption context has been set up, one of the encryption algorithms from the pool can be selected, based on a best fit strategy. Based on our experience, the pool of the encryption algorithms should satisfy the following three properties: (1) should encompass the most representative encryption algorithm families, (2) should be a minimal set of algorithms, (3) should have maximum coverage over the potential encryption scenarios.

Harnessing grid resources

Encryption operations are CPU intensive. None of the tools reviewed in the state of the art support remote parallelized encryption yet. It is a fact that using a single machine for the encryption is a performance bottleneck when dealing with large data-sets. On the other hand, data encryption is an embarrassingly parallel problem: in fact, data can be split into several chunks so that the encryption can be performed independently on each chunk. Grid Computing platforms offer a ready-to-go infrastructure where embarrassingly parallel algorithms can be run. However, privacy issues arise when data encryption is executed on a world-wide distributed environment: plain sensitive data should be processed in a trusted domain (i.e. the set of resources whose administrators are fully trusted by the end-users), whereas only encrypted data should be stored on untrusted nodes of the grid. The boundaries between the trusted and the untrusted domains are not set a priori: they can change dynamically depending on the encryption context.

Advanced key management

Generally speaking, each encrypted file has one encryption key related to it. For security reasons, it is not advisable to use the same key for different files (once the key is disclosed, all the files can be tampered). Then, the more files are encrypted the more keys are generated and need to be securely stored, thus increasing the burden of key management. Conversely, for the sake of usability, encryption key management should be transparent to the end users. Encryption tools should implement the business logic which manages the secure storage and the access to the encryption keys. A keystore server is the state of the art solution which enable users to store their keys in a remote trusted node. This brings along two benefits: (1) user can access their keys anytime/anywhere, (2) ACLs can be set to share keys access among a group of users.

However, in distributed environments keystores can be deployed throughout multiple—potentially untrusted—administrative domains. In such scenario, the Shamir's Secret Sharing algorithm can be useful, since it breaks the encryption key in n parts which are stored in separate keystores. The original key can be reconstructed by retrieving m pieces ($m < n$) from the keystores.

Design and implementation

EncryptMe is a flexible tool to encrypt and store sensitive data on untrusted remote nodes across distributed multi-domain environments. The EncryptMe design is based on the following assumptions: (a) the tool must manage both local (on the user workstation) and remote (on storage nodes) data storage, (b) the tool must support both local and remote encryption. Considering such assumptions, four data flows can be defined (Fig. 1):

- the (1) Local/Local data flow—this is supported by all the tools reviewed in the state of the art (Related work). Input data are on the user workstation; the user runs EncryptMe to encrypt data locally and then uploads the encrypted output to a storage node;
- the (2) SE/Local data flow—the user plain data are already stored on a trusted Storage Element (SE). Firstly, input data are downloaded on the user workstation where EncryptMe is running. Then, data are encrypted locally and eventually is uploaded to the target storage node sitting in an untrusted domain;
- the (3) SE/remote data flow—this is quite similar to the previous one, except for the fact that data are encrypted by a pool of remote computational resources sitting in the trusted environment, rather than locally. Thus, input data does not need to be downloaded on the user workstation. Eventually, data are transferred straightly from remote resources to the destination storage node;
- the Local/Remote (4) data flow—the input data are initially on the user workstation, and it is then transferred to remote computational resources which perform the encryption. Eventually, the encrypted data are uploaded to the destination storage node.

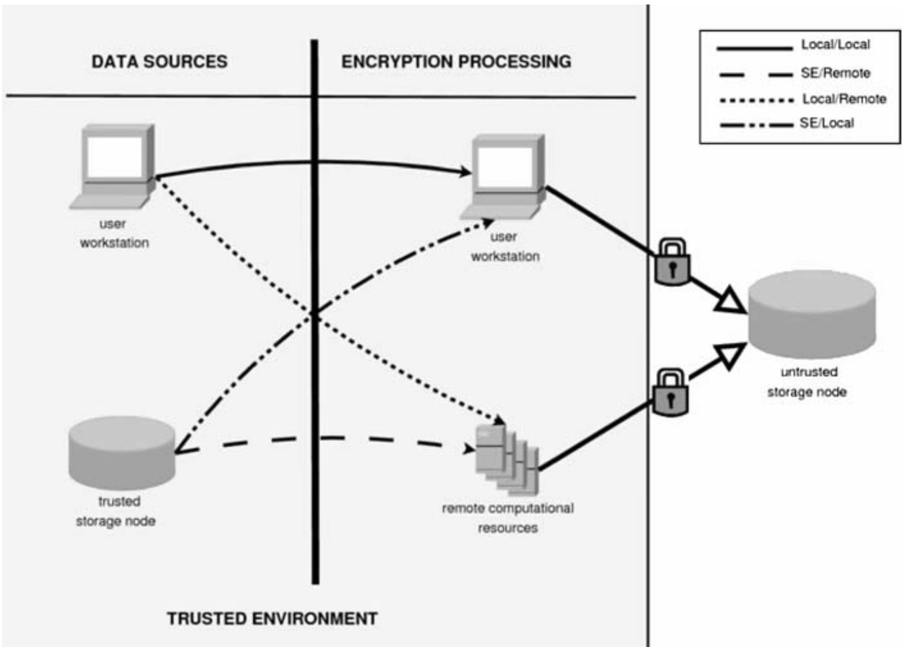

Fig. 1 System data flows

Remote parallelized encryption

EncryptMe supports remote parallelized encryption, which is not yet supported by any of the tools presented in the state of the art. There are two modules of EncryptMe involved in remote operations:

- the EM-Remote schedules, launches, monitors and stops the remote operations;
- the EM-Worker module is the one that actually performs encryption/decryption operations and it is executed on remote computational resources.

When a remote encryption operation is triggered, EM-Remote splits the input data-set in n virtual chunks. Every chunk is defined by an offset and a byte length; in addition, n tasks are created, one task per each chunk. The encryption is over when all tasks are completed successfully. EM-Remote submits m jobs. When a job is executed, it firstly (a) looks up in the table of undone tasks and gets assigned task t_i, (b) it then reads from the input location the data chunk associated with the task t_i, (c) executes the EM-Worker module to encrypt the data chunk, (d) and eventually uploads the encrypted chunk on the destination storage node. In step (a), if the undone tasks table is empty, then the job terminates. n, m parameters and chunk's lengths are dynamically calculated by EM-Remote depending on data-set size and grid resources availability and their workload. It is reasonable to assume that m should be less than or equal to n ($m \leq n$), otherwise some resources could be allocated in vain.

The mechanism described above decouples tasks and computational resources. In this way a high degree of flexibility is achieved, thus allowing faster resources to process more chunks than slower ones.

In distributed infrastructures, the time overhead related to the communication and coordination services is an important issue that needs to be taken into account. EncryptMe must deal with time overhead in case of remote encryption operations. The overhead time can't be neither controlled or annulled. EncryptMe tries to estimate the operation times for both local and remote—considering overhead—execution and, hence, the fastest method is suggested to the user. This mechanism assures that remote operations are executed only when it's time-convenient. In the "Performance evaluation" section some experimental results are reported showing when it is convenient a local encryption rather than a remote encryption and viceversa.

System components

The whole EncryptMe system consists of three main high-level components that interact with each other: (1) the user application EM-UI, (2) the EM-KeyServer and (3) the EM-Worker cryptographic module. This three components can be deployed on different nodes of the infrastructure depicted in Fig. 2.

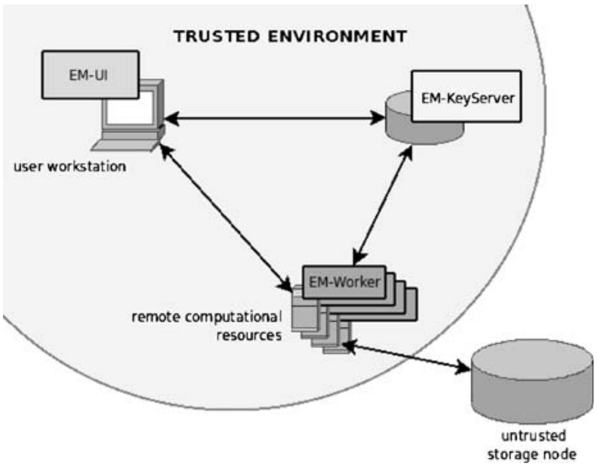

Fig. 2 System components

EM-UI

EM-UI is the main component and represents the interface of the whole system with the user. EM-UI runs on his workstation and enables the user to browse the files to be encrypted/decrypted, to launch and monitor new operations. EM-UI manages encryption process through two submodules: (1) EM-Local which is responsible for local operations and (2) EM-Remote for remote parallelized ones. Moreover, EM-UI interacts with the key server when a new file needs to be encrypted, in order to associate a new key to such file.

When a new encryption operation is submitted by the user, EM-UI interacts with EM-KeyServer that create a new encryption key for the file and stores it in its repository. Then, the newly created key is accessed by the EM-Worker module that performs the actual encryption. Furthermore, through the EM-UI module the user can define an ACL that defines the key—and, hence, the file—access policies for other users.

EM-KeyServer

The EM-KeyServer is a node used by a system to store and retrieve encryption keys. EM-KeyServer holds a binding between a key and the file encrypted with that key. By default only the owner of the file can access the key stored on the server, but it's possible to set an ACL to define the key access policies also for other users. Authentication is required to access to EM-KeyServer. Valid authentication credentials are: (1) username and password pair or (2) user certificates (e.g., X509 certificates used in the PKI).

At least one key server is required to get the system work: in this case the server needs to sit in the user's trusted environment, since that key server becomes a single-point-of-attack. Alternatively, it is also possible to distribute a pool of key servers throughout the infrastructure. In this case, the Shamir Secret Sharing algorithm allows to split the key in n pieces which are stored on separate key servers. Since the entire key is never stored on any of the key servers, such servers can also sit in untrusted environments.

EM-Worker

EM-Worker is the piece of code that actually performs cryptographic operation on input plain data. It can be either executed locally on the user workstation when a local operation is being run, or it can be executed on remote resources in case of remote parallel encryption. The EM-Worker code is not deployed on remote resources a priori, but it is sent on the fly on the resources when—and where—needed.

EM-Workers instances interact with EM-KeyServer to retrieve the keys for the operation they are currently executing. The EM-Worker module needs to be deployed in the trusted environment since it manages both plain data and encryption keys. In this way, it is ensured that encryption keys are securely managed and properly used only for the duration of the encryption process.

Implementation

The implementation of the system is in its first steps and some components and features are not implemented yet. The main programming language used is Java for his interoperability and the simplicity of deployment and execution on infrastructure nodes. EncryptMe has been implemented to work on top of grid infrastructures based on the gLite (http://glite.web.cern.ch/glite/) middleware.

The EM-Worker module implements all cryptographic primitives. Basically it receives an input stream of bytes and outputs a stream of bytes as well. It supports both local and grid file access. The grid file access is performed using GFAL (http://wiki.egee-see.org/index.php/Data_Access_with_GFAL) APIs. For the ciphers implementation included in this module the Bouncy Castle Cryptographic Libraries (http://www.bouncycastle.org/) are used (the java version).

To date, the major lack in the system is the key server that has not been implemented yet. The key management operations are entirely delegated to EncryptMe that creates the keys and stores them on the user workstation. Hence users must protect their keys on their own. Furthermore in order to share the encrypted file, the user must announce the key to other users in—potentially—insecure ways. The current implementation of EncryptMe application is quite complete. Since it is written with Java technologies, it's virtually compatible with many platforms and operating systems. It provides a GUI written in Java Swing that is structured as a typical wizard that guides the user through the input/output files selection, the setting of all operation's parameters and, eventually, the monitoring of the ongoing operation. To date, EncryptMe doesn't support grid operation by itself: all the grid operations are executed via some bash scripts described below.

Grid operations are wrapped by some bash scripts that schedule, launch, monitor and stop the grid jobs that realize the requested operation. These scripts uses the gLite bash commands for linux (e.g., `lcg-cp`, `lfc-ls`). This implies that only support gLite-based grid infrastructures are supported and that the user needs a gLite installation on his workstation (best known as gLite User Interface). These scripts should be replaced by a grid module entirely integrated in EncryptMe application (using gLite java APIs), or, at

Table 2 State of the art and EncryptMe comparison: gap analysis

Features	S3	GS3	Hydra	Perroquet	EncryptMe
Remote encryption	x	x	x	x	✓
Ciphers pool	x	x	x	x	✓
Keystore	✓	✓	✓	✓	✓
ACLs	✓	x	✓	✓	✓
Shamir Secret Sharing scheme	✓	x	✓	✓	✓

least, EncryptMe main application should interface with these scripts. In any case, users will not interact directly with these scripts.

A comparison between the features provided by EncryptMe and tools reviewed in the "Related work" section is shown in Table 2.

Performance evaluation

Algorithms pool tests

Although many investigations about ciphers performance exists (Schneier et al. 1999; Preneel 2002; Bernstein 2006; Aoki and Lipmaa 2000; http://www.schneier.com/twofish-performance.html; http://www.scribd.com/doc/4522/Comparison-between-AESRijndael-and-Serpent), it is difficult to compare such results, since each research defines its own testsuites, software implementations, hardware architectures and measure units. Performance tests have been executed on a group of candidate ciphers in order to select the subset of ciphers to be included in the reference encryption algorithms pool. The reference hardware platform for these tests is an Intel Pentium 4 @ 3.0GHz with 1GB of ram, whereas the encryption algorithms reference implementation is provided by the java Bouncy Castle (http://www.bouncycastle.org/) libraries.

The performance tests have been carried out in the following way: each cipher has been tested against 5 data-sets of different size (1MB, 10MB, 100MB, 1GB and 10GB respectively) and the total encryption time has been recorded. All the tests are based on the assumption that both input and output data are stored on the same workstation where the encryption process is executed. Tests results have been analyzed for two purposes: (a) firstly, to select a subset of ciphers to be included in the reference encryption algorithm pool and, secondly, (b) to extrapolate a performance function for each algorithm. The algorithm's performance functions are used by EncryptMe—together with other parameters—to select the best ciphers for a given operation depending on the encryption context.

According to the experimental results (Fig. 3) and the state of the art on encryption algorithms, the chosen algorithms are: (1) AES 256—because it is the worldwide standard for encryption, (2) RC6 256—because it is fairly faster than AES, (3) HC 256—because it is the fastest one, though the less secure, (4) Serpent 256—because, though it is the slowest, it is considered the most secure (Anderson et al. 2000).

Grid tests

The current implementation of EncryptMe requires users to select whether to execute the encryption locally or remotely. The next release of EncryptMe will provide user support for the decision-making process, through the estimation of the encryption time in a local or remote scenario. The latter scenario differs

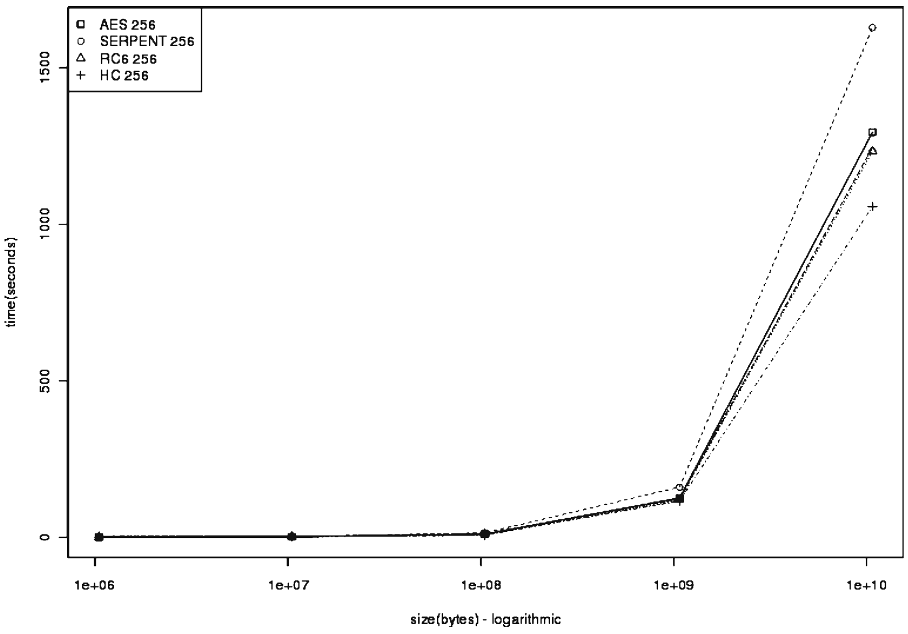

Fig. 3 Cipher tests

from the former mainly for the time overhead related to the interactions with grid resources and services. On the other hand, parallelizing the computation on several remote resources reduces the encryption time. Since the overhead time can be approximated to a constant value for all operations, the speed up factor of the operation is—partially—controlled by the number of resources that take part in the computation.

The aim of the grid tests taken is to estimate the overhead time introduced in operations by the interaction with the grid resources and services. Then it will be possible to calculate the threshold beyond that it's time-convenient to execute a remote operation instead of a local one. Since the operation time is proportional to the data-set size, it is possible to define also a size threshold: if the data-set is smaller than the threshold it's not time-convenient to execute the operation remotely.

The tests consists of seven encryption operations—that differs for the data-set size—executed both locally and remotely. The chosen data-set sizes are: 10MB, 100MB, 500MB, 1GB, 5GB, 10GB, 15GB. The input data-sets are stored on a grid storage element both for local and remote operations. Thus, with reference to the data flows presented in "Design and implementation" section, the SE/Local and the SE/Remote data flows are tested respectively. In the remote operations the size of chunks is fixed to 100MB, except for the 10MB data-set where there is only one chunk of 10MB. The number of resources allocated for the remote encryption equals the number of chunks, up

Table 3 Local and remote tests executions

Experimental results

Size	Local	Remote	
		Enc. time	Tot. time
10 MB	44	40	312
100 MB	92	88	365
500 MB	349	233	521
1 GB	694	457	734
5 GB	3221	692	941
10 GB	6235	1025	1259
15 GB	9198	1432	1723

to a maximum limit of 30 resources. If there are more chunks than resources, some resources will process more than one chunk. In tests, the encryption algorithm is fixed at AES-256 for all operations.

Local tests have been executed on a machine equivalent to an Intel Xeon CPU @ 2.0 GHz with 512MB of ram. Grid tests have been taken on GILDA (https://gilda.ct.infn.it/)—the Grid Infn Laboratory for Dissemination Activities. GILDA is a testbed grid infrastructure provided by INFN—the Italian National Institute of Nuclear Physics—which counts about 100 CPUs and 2 TB of storage space. It consists of four main computational sites (`ct.infn.it`, `cnaf.infn.it`, `sztaki.hu` and `rediris.es`) plus other minor sites in Ireland, Brazil and Argentina.

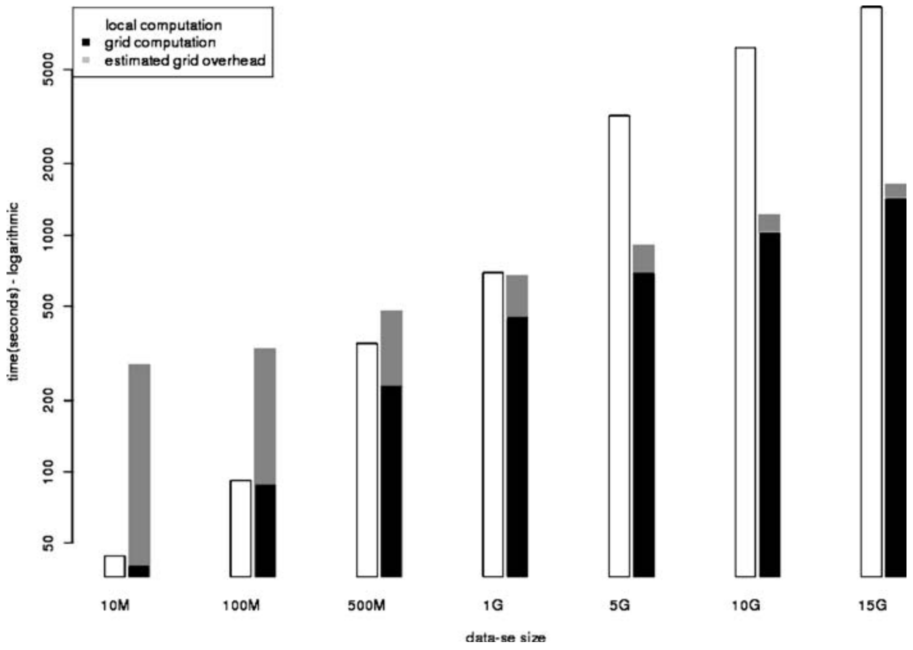

Fig. 4 Comparison between local and grid tests execution

Tests results (Table 3, Fig. 4) show that the overhead time is quite high: around 300 s. However, it is worth noting that GILDA is just a testbed, hence its performances are much lower if compared to a grid infrastructure in a production environment. 300 s is the time needed to encrypt a data-set of 500MB locally, hence for all operations that involve less than 500MB of data it is time-convenient to execute the operation locally. Conversely, in case of operations on big data-sets—more than 1GB—, it is time-convenient to execute the operation remotely because the grid overhead time is negligible compared to the total operation time.

Conclusions and future work

In this paper we presented a solution to mitigate the risks related to the storage of personal data in untrusted domains. Large-scale distributed computing has mainly focused on data security by securing the communication channels (e.g. to prevent eavesdroppers from listening at the conversation between two end-points). However, user data are stored on remote servers in plain text; even though ACLs are a common way to protect the information stored, stored data are still vulnerable to insiders. Thus, the paradoxical result is that it is more convenient to steal data by hacking the access to the storage nodes, than trying to listen to the secure conversations.

EncryptMe, the tool proposed, aims at mitigating such risks by encrypting the sensitive user information before the upload on an untrusted storage server. In addition, our tool is designed to harness the distributed computing resources available in a trusted environment, in order to speed up the encryption process. The definition of the encryption context (given in the "Challenges and motivation" section) is the key point that drives the local/remote decision making process. Performance tests have identified that 1GB is the threshold beyond which it is more convenient to switch to remote parallelized encryption. Then two scenarios can be considered: (1) local encryption for data sets smaller than 1GB (e.g. identity-related information, personal files such as documents or pictures), and (2) remote encryption for data sets larger than 1GB (users' personal backups such as entire disk partitions, or enterprise-level backups such as very large logs or user accounting information to be stored for long-term).

Future work is focussed on implementing the EM-KeyServer module to improve the encryption keys management, making it transparent to the end-users. If new algorithms or different encryption key size need to be added in the tool, it can be done very easily due to its flexible design. The 1GB threshold is the result of the tests performed on the GILDA testbed; the next steps will be oriented to testing the tool against other platforms in order to synthetize a wider spectrum of results into a set of performance indexes independent of the hardware infrastructures. Finally, cloud computing environments will be investigated in order to evaluate the adoption of encryption tools as integrated services.

Acknowledgements The work presented here has been partially funded by the DHITECH as part of the GRId For fINance project, based on the award n.DM28488 of Italian Minister of Education, University and Research. Authors especially thank the GILDA support team for the performance tests, their colleague Isabel Matranga for the proof reading work and all the Distributed Computing team at Engineering Ingegneria Informatica S.p.A. R&D laboratories for comments and suggestions.

Acronyms

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
DN	Distinguished Name
EGEE	Enabling Grid for E-sciencE
e-Id	electronic Identity
EM	EncryptMe
EM-UI	EncryptMe User Interface
GFAL	Grid File Access Library
GILDA	Grid INFN Laboratory for Dissemination Activities
GS3	Grid Secure Storage System
INFN	Istituto Nazionale di Fisica Nucleare
PKI	Public Key Infrastructure
POSIX	Portable Operating System Interface
REST	REpresentational State Transfer
S3	Secure Storage System/Service
SE	Storage Element
VO	Virtual Organization
WSDL	Web Service Description Language

References

Agir. Global analysis of radiological images—web site. 2009. http://www.aci-agir.org/.
Amazon. Amazon web services web site. 2009. http://aws.amazon.com/.
Amazon. Amazon web services security whitepaper. 2009. http://s3.amazonaws.com/aws_blog/AWS_Security_Whitepaper_2008_09.pdf.
Anderson R, et al. The case for serpent. 2000.
Aoki K, Lipmaa H. Fast implementations of AES candidates. In: In the third advanced encryption standard candidate conference. 2000. p. 106–20.
Bernstein DJ. Comparison of 256-bit stream ciphers at the beginning of 2006. In: Workshop record of SASC 2006 stream ciphers revisited, ECRYPT network of excellence in cryptology. 2006. p. 70–83.
Blanchet C, et al. Building an encrypted file system on the egee grid: application to protein sequence analysis. In: ARES '06: proceedings of the first international conference on availability, reliability and security. Washington, DC: IEEE Computer Society; 2006. p. 965–73.

Cameron K. The laws of identity. 2005. http://www.identityblog.com/?p=352.
Consorzio. Cometa web site. 2009. http://www.consorzio-cometa.it/.
Dropbox. Dropbox—web site. 2009. http://www.getdropbox.com/.
Enabling Grids for E-sciencE (EGEE). EGEE homepage. 2009. www.eu-egee.org.
Facebook. Facebook web site. 2009. http://www.facebook.com.
GFAL. GFAL web site. 2009. http://wiki.egee-see.org/index.php/Data_Access_with_GFAL.
Gilda. Gilda project web site. 2009. https://gilda.ct.infn.it/.
Glite. Glite web site. 2009. http://glite.web.cern.ch/glite/.
Google. Google web site. 2009. http://www.google.com.
GRID CT. Grid secure storage service web site. 2008. https://grid.ct.infn.it/twiki/bin/view/Main/GridSecureStorageSystem.
Hydra. Hydra web site. 2009. https://twiki.cern.ch/twiki//bin/view/EGEE/DMEDS.
Information Card. Information Card web site. 2009. http://informationcard.net/.
Kreizman G, et al. Hype cycle for identity and access management technologies. 2007.
Montagnat J, et al. Implementation of a medical data manager on top of glite services. Technical report EGEE-TR-2006-002. 2006.
Murty J. Programming amazon web services. Sebastopol: O'Reilly; 2008.
Oauth. Oauth web site. 2009. http://oauth.net/.
Openid. Openid web site. 2009. http://openid.net/.
Parrot. Parrot web site. 2009. http://www.cse.nd.edu/~ccl/software/parrot/.
Preneel B. New european schemes for signature, integrity and encryption (nessie): a status report. In: PKC '02: proceedings of the 5th international workshop on practice and theory in public key cryptosystems. London: Springer; 2002. p. 297–309.
Recordon D, Reed D. Openid 2.0: a platform for user-centric identity management. In: DIM '06: proceedings of the second ACM workshop on digital identity management. New York: ACM; 2006. p. 11–6.
Rodrigues R. Digital identity, anonymity and pseudonymity in India. 2007. http://ssrn.com/abstract=1105088.
Scardaci D, Scuderi G. A secure storage service for the glite middleware. In: IAS '07: proceedings of the third international symposium on information assurance and security. Washington, DC: IEEE Computer Society; 2007. p. 261–6.
Schneier B. Why "anonymous" data sometimes isn't. 2009. http://www.schneier.com/essay-200.html/.
Schneier B, et al. Performance comparison of the aes submissions. In: In proceedings of the second AES candidate conference; 1999. p. 15–34.
Scribd. Comparison between aes-rijndael and serpent. 2009. http://www.scribd.com/doc/4522/Comparison-between-AESRijndael-and-Serpent.
Shamir A. How to share a secret. Commun ACM 1979;22(11):612–3.
Shibboleth. Shibboleth web site. 2009. http://shibboleth.internet2.edu/.
Spideroak. Spideroak—web site. 2009. http://www.spideroak.com/.
The Legion of the Bouncy Castle. Bouncy castle crypto apis. 2009. http://www.bouncycastle.org/.
Twofish. Performance vs. other block ciphers (on a pentium). 2009. http://www.schneier.com/twofish-performance.html.
Yahoo. Yahoo web site. 2009. http://www.yahoo.com.