Paillier-Based Protocols for Encrypted Live Collaborative Text Editing

Wilton Cappel
wcappel@oxy.edu
Occidental College

1 Introduction and Problem Context

Collaborative cloud document editing has become a staple technology both in professional and personal domains over the past decade and a half. However, current popular solutions rely on storing a document insecurely on the cloud—where the host has access to the text contents and potentially personally identifying information. The consequences of this are the possible scenarios of data leakage, data exfiltration by insiders, or inappropriate usage of information by the host [1]. In terms of accountability, cloud computing is also a relatively more difficult area to legislate, due to its internationality posing a challenge for how current laws are based within national jurisdictions [12].

In order to provide an alternative approach that both obscures the contents of the text and represents the document in its current state, we utilize the Paillier partially homomorphic encryption scheme to develop and analyze privacy-preserving live collaborative text editing protocols; we additionally develop a variant of the initial protocol that attempts to obscure the editing positions of the users in the document.

2 Technical Background

Collaborative cloud text editing can be generally described as users making concurrent edits to a text data structure hosted on a server. In order to resolve conflicts between concurrent edits, there are two main approaches of operational transformation algorithms (OT) and conflict-free replicated data types (CRDT). Operational transformation reduces user actions into operations, which are submitted to a server by clients with a live connection to it; operations are transformed by the server to include changes that have happened since their transmission while preserving user intention—these are then applied to the document state and rebroadcasted back to clients [9]. In contrast, CRDTs are data structures that are designed specifically to be collaborative without necessarily relying on a central server. This is done by guaranteeing the correctness of changes even when they are divergent in terms of the state of the document [17]; at any given point in time there is no single authoritative state for a document until clients merge their views.

Homomorphic encryption (HE) is a type of encryption that allows accurate computation to be performed on encrypted data without decrypting or revealing the secret key. The concept is based on the homomorphism property, which allows for a function between two algebraic structures of the same type that preserve their operations. Within the umbrella of homomorphic encryption itself, there are different levels, such as fully homomorphic encryption and partially homomorphic encryption. Partially homomorphic encryption only either only one type of or a limited number of operations on encrypted data, while fully homomorphic encryption allows any number of addition and multiplication operations.

The Paillier cryptosystem is an additively partially homomorphic public-key scheme—in which a publicly-available key is used for encryption while a private, secret key is used for decryption. Paillier's intractability (or "hardness assumption") is based on decisional composite residuosity [13]. An instance of a Paillier secret key is made up of two large, primes equal in bit length. The product of these two primes is known as n and makes up one component of the public key [4].

Message authentication codes (MACs) are mechanisms for providing guarantees on the authenticity of communications against tampering by adversaries. MACs are composed of three algorithms: key generation, signing, and verification [11]. The secret key k (shared between parties who wish to communicate securely) acquired via the key generation algorithm is fed alongside a message m into the signing algorithm, producing a tag t. Upon receiving message m and tag t, the integrity of the message can be ensured by using the verification algorithm on the values of k, m, and t to validate that the message matches the tag. With a secure MAC, an attacker should never be able to forge a valid tag on a message not authorized by the sender. MACs are often used alongside encryption schemes resistant to chosen-plaintext attacks to mitigate against the higher attack model of chosen-ciphertext attacks.

3 Prior Work

One previous implementation of a privacy-preserving cloud text editing project is SecureCEdit, which is an end-to-end encrypted solution built on top of Google Drive [3]. SecureCEdit works by encrypting a document's contents and then storing it on Google Drive. With added key exchange, it allows for the possibility of collaboration between users. However, every time the cloud state of the document needs to be updated, the entire document must be re-encrypted, not immediately reflecting its live state. Another similar project is Content Cloaking, a client-side solution that is focused on low-software requirements, also working on top of cloud document providers [5]. Content Cloaking works on the level of AJAX to ensure that messages are symmetrically encrypted when sent and decrypted on receipt to store text content on a server. However, it necessitates re-encryption for modifying any text content on the server.

Another work focused on providing a solution for private, secure cloud document editing is by Yeh et al., in which they utilize a red-black tree to organize the document and encrypt blocks of data separately so that the full document does not need to be repeatedly decrypted and encrypted upon changes occurring to it [18]. They find that although using such a data structure may incur greater computational cost, for the operations of encryption/decryption, insertion, and removal it is more efficient. One very recent project concerning private collaborative document editing is Skiff Pages, an application for end-to-end encrypted document editing. Skiff uses client-side CRDTs with the server as a public channel to pass end-to-end encrypted updates between users that they merge into their respective CRDTs; upon finishing an editing session, the text content in the CRDT is encrypted and uploaded to the server [16]. Another interesting work is by Arora and Atrey, in which they utilize secret sharing rather than a scheme like AES to achieve greater efficiency as well as keyword-based search on a private document [2].

One recent application of the Paillier scheme for cloud computing is for privacy-preserving outsourced image editing, by Aniket Das et al. They extend the Paillier scheme for operation over images, specifically in the form of a matrix of pixel grayscale values; they then define an image brightening function based on homomorphic additon and multiplication that the server can perform without knowledge of the image data [6].

4 Methods

4.1 Text Representation

We decide to leverage homomorphic encryption in our solution for the benefit of maintaining a live state of the document on the server at all times without continual re-encryption of entire contents or updating server-side only at the termination of a collaborative session. In order to utilize the Paillier scheme for working with text data, we can first divide a text document into fixed-size chunks of bits based on the byte encoding representation of the characters (UTF-8, ASCII); each chunk of text is represented as the concatenation of the character bytes into a binary number converted into an integer value, which is now compatible with the scheme. The bit length of each chunk has been provisionally chosen for the purposes of the project to be 64, as this allows for representation in UTF-8 of up to 8 byte-length characters or 2 max-size characters. The decision to represent text in chunks for use with the scheme rather than on a character basis is due to considerations of storage and computational efficiency when considering the ciphertext expansion of the scheme. Assuming a Paillier key size of 2048 bits and a standard character being 8 bits in length, the ciphertext expansion of encrypting a single character would lead to an increase in bit length by a factor of 32; as a ciphertext's bit size is double that of the key [4]. However, while packing characters into chunks lessens ciphertext expansion and storage requirements, it can also reduce editing flexibility due to how it increases the "region size" for chunk operations in the text document, as edits on a single chunk will cover more area of the document. In finding an optimal chunk size, both of these factors need to be balanced. Because of this integer encoding scheme, text edits can be transformed in terms of arithmetic operations on the value of a chunk based on the difference between the integer values in the character encoding scheme between the initial chunk and its new edited state.

4.2 Initial Protocols

To actively manipulate text, a data structure such as a linked list can be used in which each node contains the integer value of a chunk and points to the next chunk in the document. Text editing operations can then be defined as either adding a new chunk node at an index in the list, adding to the value of a node at an index in the list, or removing a node at an index in the list. For hosting and editing the document on the cloud, clients and the server can have a plaintext and ciphertext version of the data structure respectively. To perform edits, clients submit operations to the server over a channel with the encrypted value of the integer chunk representation, specifying both the index in the list and the type of operation; after the server validates the operation by performing it on its encrypted version of the data structure, it rebroadcasts the operation back to all clients—only then is the change visibly reflected in their local document. In addition to validation and rebroadcasting, the server can also run operational transforms on clients that submit operations that lag behind the current state of the document by utilizing a revision number that increments with each operation performed on the document; upon receiving a lagging oper-

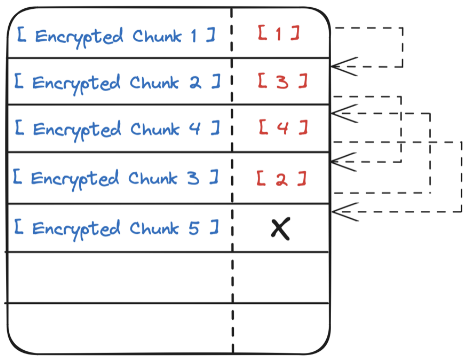

Figure 1. Obfuscated data structure variant

ation, the server can "catch it up" to be applied on the current document by transforming its index or addend value (in relation to the latest operations) to preserve the intent of the user. Currently with the scheme, this allows for incorporating changes in chunk indices as well as the possibility of a last editor wins strategy, in which the intention of a lagging user can be preserved.

During document creation, a user generates a Paillier keyset, sends the public key to the server and stores their private key locally. Collaborators can be securely invited to the document by using a temporary symmetric encryption scheme established via key exchange to share the Paillier secret key for ciphertext encryption, over the public channel of the server. Upon entering into an editing session, a client downloads the document from the server and decrypts it into their plaintext document, and waits to receive updates.

4.3 Obfuscated Variant

While this approach obscures the text content of the document, it does not obscure the editing indices of operations on the document nor the type of those operations. To overcome these limitations, an obfuscated variant of the linked list data structure is proposed. The difference from the initial variant is in how it is instead represented as an array of tuples in a table structure rather than with nodes. Each tuple is composed of both the value of the integer chunk and a "next" pointer which describes the table index location of the next chunk of text. Fig. 1 displays a visual representation of the data structure. The document is initialized by creating a configurable size of tuples which all have their chunk value set to an encrypted 0 value, and their next pointer value set to an encrypted non-index value such as -1. As a result, editing operations are transformed to reference table indices as well as the multiple values in tuples. Due to the need to preserve the order of pointers in the table, add and remove chunk operations need to be defined as a transaction of multiple operations: addition on the primary tuple's chunk value and addition on the previous tuple's index pointer. As a result, the order of tuples in the table does not need to be linear, as long as the pointers referring to the order of chunks in the document are preserved. Adding and removing chunks as a transaction also appears to take the same format and in order to discern between the two, the server needs prior information. Since operations utilize homomorphic addition, this also opens up the possibilities of masking edit operations as add/remove operations, or the ability to entirely fake operations—both by replacing any operation addends with an encrypted 0 value.

4.4 MAC Incorporation

Due to malleability concerns discussed in detail in the section below, there is the option of incorporating a MAC into the document by sharing a MAC secret key using the temporary symmetric scheme established for collaboration. Operations incorporate MACs by tying it to the ciphertext content, allowing for clients to validate that the decrypted content of rebroadcasted operations matches the tag using the MAC key. The drawback of this is that it does not allow for operational transforms on the text content of operations. In order to ensure the server has not tampered with the document outside of the operations in the editing session, a hash based on both the secret key and the contents of the document can be updated on the server by the last client to leave the session. Upon decrypting the document, clients can check that the last hash matches the current contents of the document. Another idea that was considered in place of a MAC was to "seed" the zero values of the ciphertext in the obfuscated form, so that the server could not modify the ciphertexts it knows are zero-valued; however, this still does not alert the user if the server tampered with the document, even if the tampering does not lead to a necessarily legible result.

5 Evaluation and Security Analysis

5.1 Metrics

In order to demonstrate that the Paillier scheme is appropriate for the use case of text editing in terms of speed and responsiveness, benchmark tests for the operations of encryption, decryption, and homomorphic addition will be performed in which an average elapsed time is taken over 100 instances with randomized 64 bit plaintext and corresponding ciphertext values; additionally, the difference in speed will be examined between scheme key sizes of 2048 and 3072 bits to check for a relationship. These are relatively informal tests to demonstrate that the operations "feel" fast enough for the use-case and will be put into perspective using metrics from other domains concerning response times. It should also be noted that the results of these benchmarks are influenced by hardware capabilities and may vary between computers as well as with the implementation of the scheme.

In looking at the security properties of Paillier and translating these into the project's application of the scheme, the conventional attack models of chosen plaintext and chosen ciphertext will be applied alongside examination of the scheme's intentional property of malleability. Concerning malleability, we discuss how MAC integration can serve to mitigate this vulnerability for the purposes of document editing.

5.2 Security Analysis

As a public-key scheme, Paillier has the property of non-deterministic encryption, randomness is incorporated when encrypting plaintext values into ciphertexts. Because of this, plaintexts do not have a single corresponding ciphertext value. As a result, this renders chosen plaintext attacks infeasible since it would be equivalent to solving the underlying hardness assumption of the scheme [13]. However, Paillier's inherent malleability (the ability to transform a ciphertext into another related ciphertext without the secret key) opens up the scheme to chosen-ciphertext attacks, in which an adversary can gain information about the secret key by receiving decryptions of ciphertexts they have chosen. Assume an adversary is given both encryption and decryption oracles, a ciphertext of their choice c, and can decrypt any ciphertext other than c. If the adversary encrypts 0 and adds it to c to result in c', decrypting c' gives them the equivalent plaintext of c. Consequently, Paillier is not secure against CCA due to its homomorphism property which the attacker can exploit to learn information about the secret key. The significance of this is that if the attacker has access to decryption capabilities (by unknowingly deceiving a client), the scheme is no longer secure.

The malleability of the scheme that results from its property of homomorphism also allows the server to tamper with the values of both operations submitted by clients as well as the encrypted version of the document. In order to circumvent this, at least to the extent that clients will be able to definitively know if tampering has happened, MAC integration can be used to invalidate a ciphertext if the verification algorithm finds that the tag and plaintext do not match (as the tag is generated using the MAC secret key, which the server does not have access to). The encrypted document can be verified against tampering outside of a session using hashing and a secret key (which was explained in the methodology section).

Key Size	Encryption	Decryption	H. Addition
2048	0.0102	0.0022	5.44999e-06
3072	0.0348	0.0079	9.02335e-06

Figure 2. Avg. time in seconds for scheme operations

5.3 Obfuscated Variant Test Suite

We look at the perspective of a server tracking positional information of operations, as well as see how this changes with the utilization of fake and masked operations. In addition, two types of editing patterns are used both with and without the assumption of fake edits: sequential addition of new chunks interspersed with edits on existing chunks, and randomized add and remove operations alongside edits. In order to examine these scenarios, a test suite that sets up a randomized editing script that tracks information according to an adversary's perspective is used.

To analyze the effectiveness of obscuring editing locations in the obfuscated variant, we look at how the distinguishability of add/remove chunk operations based only on prior information affects how the server tracks positional information. The server can always assume that the first operation submitted is to add the initial chunk; it can then proceed to determine whether subsequent transactions (composed of two operations) are chunk additions or removals if one of the operations is on a tuple that has not been used before in table (or was previously removed) or not, respectively. If there is allowance for fake add/remove operations, then this becomes a harder task for the server as it has to either assume the transaction is genuine or not; edit operations also need to be scrutinized as well due to how they can be masked as add/remove transactions by incorporating a fake operation. This is implemented in the test suite as the adversary exhaustively branching on the state of the document per transaction between genuine or not, while still being informed of what chunks have been used (this is done by using a series of linked lists to attempt to mirror each possible branch); the relative estimated range of an operation's edit position can be seen as its minimum and maximum position in all possible branches. If there is a "goal" that needs to be achieved, it is to render the estimated position range for an operation in the document to be as wide as possible.

6 Results and Discussion

In Fig. 2 are the measured average times in seconds for the scheme operations on 64 bit integer values. Unsurprisingly, homomorphic addition is by far the fastest operation due to how it is simply defined as the product of the two ciphertexts over the modulus of n^2 [4]. Due to this, homomorphic addition can be considered as relatively negligible in terms of runtime. In comparison, encryption and decryption are relatively slower operations; encryption requires exponentation by the power of n on a random number, and decryption requires two instances of exponentiation by the power of a number half of the bit length of n (one of the primes used in key generation) [4]. However, the slowest time out of the results at around 35 milliseconds can still be deemed responsive enough for text editing—yet this could be compounded by added network latency in a practical setting.

After running the test suite scenarios on the obfuscated variant, it is clear that under the assumption of honest/genuine operations submitted by clients, the server is invariably able to track the absolute positions of edits. However, if the server cannot assume that operations are indeed genuine and not just containing zero-valued addends, then it is more difficult to discern the absolute positions of operations to the point where it needs to estimate them in terms of ranges. Additionally, this causes the observing server to be more unclear about whether an operation is an addition/removal, as prior information can become contradictory due the need to account for whether it is genuine or not. Given a sufficient amount of operations that the server is uncertain of are real or not, the estimated range of an operation's position in the document could become large enough to be completely non-specific in terms of location (in some cases, representing the width of the document itself).

However, the caveat of incorporating fake operations is that the server must not be able to discern them—so the client must have a capable mechanism of continuously generating them to where even on a metadata (timestamp) level, the server cannot distinguish them. Another concern is that this could make editing much more computationally expensive to the point where it may not even be worth it.

Future development would examine the creation of a good mechanism for faking edits, an authoritative measure on an optimal chunk size to be used with the scheme, as well as running benchmarks on MAC and hash related operations to determine how it would affect the responsiveness of collaborative editing sessions. In addition, seeing how it practically works in a server environment could help it be tested with average document sizes to determine responsiveness when downloading documents and hashing.

7 Ethical Considerations

7.1 Energy Efficiency and User Experience

Due to heavier computational overhead, using a partially homomorphic scheme such as Paillier could increase energy requirements and computational complexity for using cloud text editing applications. This has the effect of both increasing emissions and potentially degrading the experiences of those who have less capable devices—as privacy should not just be a luxury for those who can afford better hardware. Roussev and Vassil suggest that collaborative editing systems should have options for users to choose how they can interact with networked collaborative models, in terms of how their edits are operationally resolved and interact with the server as well as other collaborators [15]. Further concerning user experience, Dishaw et al. suggest that an important factor in the success of collaboration tools is the clarity of the collaboration process [10]; this factors into how the user can be implicitly informed of how the operational model works and the relationship they have to the server and their peers.

7.2 Accountability and Transparency

While stringent privacy regulations on cloud computing are beneficial for consumers, they may prevent users from being able to dictate their own desired policies in how might their data be used, as privacy tends to also be a subjective, individual matter [1]. Rather than dictate policy from the top-down, users should be empowered to make their own choices by developing habits based on being informed about their engagement with cloud services; as the most widely accepted definition of privacy in security circles is informational self-determination: the ability for individuals to self-determine what information is communicated about them to who and how [8]. As a result, cloud computing should shift its focus to empowering the user by giving them further control over their data and informing them further, rather than letting them blindly trust the service provider. Another related idea is to allow users to define their own cloud accountability requirements and service agreements; users are able to enforce their policies and determine whether the service provider breached them via tools based on a detective methodology, which could become possible if the abstraction layers within cloud services were distinctly separated and could be audited [1]. Another related concern is the regulation of when data should be ”live” on a cloud computing platform; a good heuristic is that data should only be live only when the user is, and should be linked in activity to the direct input of the user, tied directly to their decisions about how it should be stored. In relation to this, the technical infrastructure that stores and transmits user data has become increasingly obscure to the point where individuals may have no idea of who has their data at all [14].

7.3 Power

In providing digital infrastructure for a much broader selection of people, cloud computing unfortunately results in the aggregation of vast swathes of data into a handful of cloud data providers, who tend to be large, wealthy tech companies that can afford the infrastructure for hosting cloud services. As a result, this can lead towards the creation of an oligopolistic market due to how they exert influence in regards to setting prices and determining which kind of services are adopted into the market. These companies also ultimately set the standards for end user agreements, and users are forced to accept them in order to benefit from their provided services [12]. Trust is established between a user and the service provider through the service agreement; however, it is usually a one-way dialogue with the service provider dictating the terms without input from the client. Even if the client is able to have some say, they may not be invested in reading the terms of the agreement—assuming it is boilerplate, and so there is not an understanding of what can actually be done with the user's provided information [14].

8 Appendix

8.1 Replication Instructions

In order to replicate the project, ensure that either Xcode or the Swift toolchain is installed. Clone the repository with `git clone https://github.com/wcappel/PaillierDocs` and run `swift build` in the generated directory to download package dependencies and build the project. Upon a completed build, run `swift run` to execute the `main.swift` file in the project.

8.2 Code Architecture

The code is mostly split between 4 main directories: ”Paillier”, ”Structures”, ”DocumentModels”, and ”Test Suite”. ”Paillier” contains an implementation of the Paillier scheme copied/adapted into Swift from the python-paillier library [7] (as per GPLv3, the license has been copied over) which relies on a fork of BignumGMP for Swift. ”Structures” contains the linked list and tuple variant data structures that are compatible with the scheme and its additive operations. ”DocumentModels” contains the text document implementations of the linked list variant and obfuscated variant respectively. ”Test Suite” contains the test suite implementation used for evaluating the obfuscated variant. In addition, there is the main executable file as well as a helper file for text to integer encoding.

References

[1] Yunusa Simpa Abdulsalam and Mustapha Hedabou. “Security and Privacy in Cloud Computing: Technical Review”. In: *Future Internet* 14.1 (2022). ISSN: 1999-5903. DOI: `10.3390/fi14010011`. URL: `https://www.mdpi.com/1999-5903/14/1/11`.

[2] Shashank Arora and Pradeep K. Atrey. “Secure Collaborative Editing Using Secret Sharing”. In: *2021 IEEE International Workshop on Information Forensics and Security (WIFS)*. 2021, pp. 1–6. DOI: `10.1109/WIFS53200.2021.9648395`.

[3] Shashank Arora et al. “SecureCEdit: An approach for secure cloud-based document editing”. In: Oct. 2016, pp. 561–564. DOI: `10.1109/CNS.2016.7860548`.

[4] Nicholas Chen. “A Comparison of El Gamal and Paillier Cryptosystems”. In: 2018. URL: `https://api.semanticscholar.org/CorpusID:198921184`.

[5] Gabriele D'Angelo, Fabio Vitali, and Stefano Zacchiroli. “Content Cloaking: Preserving Privacy with Google Docs and other Web Applications”. In: Mar. 2010. DOI: `10.1145/1774088.1774259`.

[6] Aniket Das et al. “Secure Outsourcing of Image Editing Based on Homomorphic Encryption”. In: *Internet of Things and Its Applications*. Ed. by Keshav Dahal et al. Singapore: Springer Nature Singapore, 2022, pp. 461–472. ISBN: 978-981-16-7637-6.

[7] CSIRO's Data61. *Python Paillier Library*. `https://github.com/data61/python-paillier`. 2013.

[8] Josep Domingo-Ferrer et al. “Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges”. In: *Computer Communications* 140-141 (2019), pp. 38–60. ISSN: 0140-3664. DOI: `https://doi.org/10.1016/j.comcom.2019.04.011`. URL: `https://www.sciencedirect.com/science/article/pii/S0140366418310740`.

[9] Andrew Herron. *Building real-time collaboration applications: OT VS CRDT*. Jan. 2020. URL: `https://www.tiny.cloud/blog/real-time-collaboration-ot-vs-crdt/`.

[10] Jakob Iversen et al. “An Examination of the Characteristics Impacting Collaborative Tool Efficacy: The Uncanny Valley of Collaborative Tools”. In: *INFORMATION TECHNOLOGY EDUCATION* 12 (Oct. 2013), pp. 301–325. DOI: `10.28945/1899`.

[11] R. Jueneman, S. Matyas, and C. Meyer. “Message authentication”. In: *IEEE Communications Magazine* 23.9 (1985), pp. 29–40. DOI: `10.1109/MCOM.1985.1092643`.

[12] Brid Murphy and Marta Rocchi. “Ethics and Cloud Computing”. In: *Data Privacy and Trust in Cloud Computing: Building trust in the cloud through assurance and accountability*. Ed. by Theo Lynn et al. Cham: Springer International Publishing, 2021, pp. 105–128. ISBN: 978-3-030-54660-1. DOI: 10.1007/978-3-030-54660-1_6. URL: https://doi.org/10.1007/978-3-030-54660-1_6.

[13] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology — EUROCRYPT ’99*. Ed. by Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238. ISBN: 978-3-540-48910-8.

[14] Luciana Rogers Duranti. “Ethics in the Cloud”. In: *Journal of Contemporary Archival Studies* 4 (2017).

[15] Vassil Roussev and Prasun Dewan. “Supporting High Coupling and User-Interface Flexibility”. In: *ECSCW 2005*. Ed. by Hans Gellersen et al. Dordrecht: Springer Netherlands, 2005, pp. 45–64. ISBN: 978-1-4020-4023-8.

[16] Skiff. In: *Skiff: End-to-end encrypted, Privacy-First Workspace* (Feb. 2023). URL: https://skiff-org.github.io/whitepaper/Skiff_Whitepaper_2023.pdf.

[17] Matthew Weidner. *Designing data structures for collaborative apps*. Feb. 2022. URL: https://mattweidner.com/2022/02/10/collaborative-data-design.html.

[18] Sheng-Cheng Yeh et al. “A study on the data privacy and operation performance for cloud collaborative editing systems”. In: *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. 2012, pp. 591–596. DOI: 10.1109/CloudCom.2012.6427609.