SecGOD
Google Docs: Now I Feel Safer!

Antonis Michalas
Athens Information Technology, Athens, Greece
and
Aalborg University, Denmark
Email: amic@ait.edu.gr

Menelaos Bakopoulos
Athens Information Technology, Athens, Greece
and
Aalborg University, Denmark
Email: mbak@ait.edu.gr

***Abstract*—This paper presents SecGOD. A tool that protects the privacy of documents created with online office suites. SecGOD is implemented as a Greasemonkey java-script making it deployable on all popular greesemonkey compatible browsers and utilizes symmetric key encryption. All operations run on the client side, with SecGOD operating invisibly as concerned by the cloud, with no changes needed to the code that is provided to the cloud server provider. Finally, the effectiveness of SecGOD is demonstrated by conducting extensive experiments measuring the processing time for the three versions of AES (128, 192, 256 bits).**

***Index Terms*—Privacy, Security, Software as a Service, Cloud Services, Cloud Computing**

I. Introduction

In the last years, cloud computing has received a great deal of attention, not only through the research community but among individual users and companies as well. Generally speaking, cloud computing is a subscription-based service where users can obtain networked storage space and computer resources. Related to the networked storage space, cloud computing is similar to an e-mail IMAP client with the difference that you can store and access any kind of information you want. Essentially this means that files on the cloud can be accessed from any location, any time and from almost any device which does not have to belong to you.

The main advantage of cloud computing, is the fact that companies can greatly reduce IT costs since infrastructure as well as storage and processing is outsourced and can be quickly extended by the cloud without needing investments in local costly hardware and installation. Despite the fact that companies can offload data and computations to cloud services, most of them hesitate to trust such services due to outstanding security concerns.

It is a fact that in November 2011 dropbox a well known cloud storage provider had a security flaw allowing unauthorized users to view any user's cloud based documents without entering a correct password [1]. Additionally, further security failures such as 6 million linkedin stolen passwords in 2012 and security issues from products by facebook and other corporations prove that security in many large web2.0 services is not as infallible as it may seem [2].

Nevertheless, major players in the technology industry have been predicting that cloud computing is here to stay, and will only be growing in the future. With cloud technology improving upon performance and security, it is clear that other industries are seeing this change and will most likely follow suit in heading to the cloud.

According to IBM, the demand for cloud computing is on the rise as organizations look to expand the impact of IT to deliver innovative services while realizing significant economies of scale. According to market research firm IDC, $17 billion were spent on cloud-related technologies, hardware and software in 2009. IDC analysts expect that will grow to $45 billion by 2013. Furthermore, in December 2011 Gartner and IDC released their latest cloud computing statistics and predictions for 2012 through 2016 [3]. At a glance, their results can be summarized in the following:

- In 2012, 80% of new commercial enterprise apps will be deployed on cloud platforms
- By 2016, 40% of enterprises will make proof of independent security testing a precondition for using any type of cloud service
- At year-end 2016, more than 50% of Global 1000 companies will have stored customer-sensitive data in the public cloud
- Mobile software as a service (SaaS) Market will reach $1.2 billion in 2011 and grow to $3.7 billion by 2016

As cloud computing grows, many IT enterprises are looking for ways to reduce expenses related to maintaining in house IT infrastructure (servers, etc) by relocating applications and data storage to the cloud. Moreover, enterprise information technology paradigms such as cloud computing, metadata search, SaaS and online office suites have gained credence as Web-based business models and operational practices have been adopted by many businesses. This has given rise to services like Google Apps and Office 365, which offer e-mail, calendaring, and other Web-based services that completely replace not just software running on company's servers, but also software running on employer's desktops.

While these applications are not as complex or comprehensive as the leading desktop counterparts, they have other advantages over traditional software. The most obvious of these advantages is that the applications are not tied to a specific computer and there is no need to download and install

978-1-908320-08/7/$25.00©2012 IEEE

software on a particular machine. Any computer that has access to the Internet can take advantage of the functionality that these applications provide. As each user saves information to the cloud system, (s)he can access the same file(s) from anywhere. In addition to that, cloud applications let multiple users to edit the same file(s) at the same time. This is called online or multi-user collaboration, and it could streamline teamwork over the Web.

One of the most popular online office suites is offered by Google under the name Google Docs. In September 2011, for example, they had over 4.4 million unique users. This number is growing even as competitors such as Zoho and Microsoft ramp up their products. Despite the great convenience of the service, many users are skeptical on using it, since they worry that their private data can be exposed to people who should not have access.

A. Our Contribution

In this work we present a tool that protects the sensitive data of users that make use of online office suites. The main goal of our approach is to protect the documents that each user creates. The main objective of our approach is to build a system that successfully protects the contents of documents created by users of these applications. To this end, we use existing techniques from the field of cryptography to encrypt the content of each document before it is transferred from the client to the server. Moreover, we manage to avoid storing the plaintext of the documents in the cloud, which protects the contents even from a possibly malicious cloud service. In addition to that, we add a functionality that is missing from the previous works in the area, we propose a technique that sharing of a document can be done in a more secure manner. Finally, the whole analysis is coupled with extensive experimental results that demonstrate the protocol's validity and efficiency over previous works in the area.

B. Organization

In Section II, we review some of the most important secure storage systems for online office suites. In Section III, we formally describe the problem, we define the basic terms that we use in the rest of the paper and we describe the threat model that this paper deals with. In Section IV we present our protocol and in Section V we provide a security discussion in which we show the resistance of our protocol in the presence of malicious adversaries. In Section VI we present experimental evidence that shows the effectiveness of our protocol while in Secction VII we conclude the paper.

II. Related Work

Although online office suites meet a significant growth and the number of users is increasing rapidly, little work has been done towards the direction of protecting legitimate users from exposing private data to unauthorized users. In this section we present the most important works regarding the security of cloud services in the particular field of online office suites.

To the best of our knowledge, M. Christodorescu in [4] is the first who introduced the problem of securing the data that users upload to remote services, such as cloud services, which may be under control from potentially malicious parties. More precisely, authors argue that the assurance of online data privacy must go beyond legal or social contracts, which provide only *post-facto* redress, to employ technical solutions. Furthermore, they sketch an architecture that enhances the privacy of data sent to remote web servers, in spite of any actions by the parties controlling the web servers. Towards this end, they proposed the use of client-side opportunistic encryption and decryption to allow the user to control the future use of any data they upload to a web server. More precisely, in order to preserve secrecy of the data after it is entered by the user, a cryptographic layer encrypts the data as the client-side web application sends to the provider and decrypts when the data travels the reverse route. Apart from that, they propose to generate a unique key, different from the password that authenticates the user. To do this, they rely on the PwdHash project [5] where a Pseudo Random function [6] is used to compute it from the password of the user. The main problem with the presented protocol, is the fact that the description is in a high level and without any kind of implementation. This makes it difficult to find any vulnerabilities while at the same time there are parts that are not clear at all. For example, authors mention that the implementation of their architecture have no impact on the programming model used be the web-application provider, but this is not clear from the provided description.

In [7] authors conceived a new transparent user layer for Google Docs, and implemented it as a Firefox add-on, which encrypts the information before storing it on Google servers; making virtually impossible to get access to the information without the right password. The main drawback, is the fact that the add-on uses two hidden documents created using the Google Docs API, which contain all the information needed to encrypt and decrypt user's information. One of the documents contains the data about the user's ciphered documents (algorithm used, password and encryption options if it is necessary). The other one maintains the same information, but only about the documents that are currently being shared.

Another approach by D'Angelo *et al.* [8] utilizes an extension for Firefox based on the gDocsBar add-on [9] called SeGoDoc. The original gDocsBar which deals with drag and drop uploading to google docs has been extended to include intermediate encryption and decryption in order to protect online document privacy and integrity against service providers. The plugin is not directly integrated with the Google Docs interface and depends on third-party developers, meaning that it is unstable and not completely transparent for Google Docs users. Moreover, the password used to encrypt the documents is saved locally, at the client side, and it currently does not support encrypted document sharing.

The most concrete work is presented by Y. Huang and D. Evans in [10]. Authors presented a protocol that enables users to use a cloud application to manage their documents

without sacrificing confidentiality or integrity. In addition to that, they used an incremental encryption scheme and extend it to support variable-length blocks. Furthermore, they provided a security analysis coupled with extensive experimental results that showed that their protocol preserves most of the cloud applications functionality with less than 10% overhead for typical use.

All of the above mentioned works, have two common elements. First, they do not propose a way with which users that share a document can securely retrieve the decryption ley. Second, for the saving procedure they absolutely rely on Google's functionality, which can be very tricky in the sense of computation since Google is using an algorithm to autosave only the changes that have been made to a document after a time interval t. This seems to be well treated only from the protocol proposed in [10] where the content of the document is divided into blocks. But still this technique is computational heavier than the one we propose in this paper.

III. Problem Statement & Adversarial Model

We start by providing a definition of online office suites as described in [11].

Definition 1: An online office suite or online productivity suite is a type of office suite offered by websites in the form of SaaS. They can be accessed online from any Internet-enabled device running any operating system. This allows people to work together worldwide and at any time, thereby leading to international web-based collaboration and virtual teamwork. The packages usually includes: word processing, spreadsheets, presentation, email, and database. Usually, the basic versions are offered for free and for more advanced versions one is required to pay a nominal subscription fee.

Problem Statement: *Let $U = \{u_1, \ldots, u_n\}$ be the set of all users that uses an online office suite. User u_i wants to create a document d_i and ensure the secrecy of its content by making it impossible for any user without privileges to access the enclosed information. Furthermore, since online office suites offer the option of sharing d_i with other users, it is likely that u_i will want to share d_i with users from a set $U_{d_i} \subset U$, $u_i \notin U_{d_i}$. In this case, u_i must share a secret with all users in U_{d_i} in order for each one to be able to access the content of d_i. So, the problem is to find a way to encrypt d_i and save (send to server) the encrypted content instead of the plaintext. Additionally, each user that belongs to U_{d_i} (including also the creator u_i) must know how to effectively decrypt it in order to access its content.*

For the needs of our protocol, we assume that the reader is familiar with the basic concepts of cryptography.

A. Adversarial Model

Similar to the work presented in [10], the protocol in this paper assumes that the adversary has computational power equivalent to a probabilistic polynomial time Turing machine that fully controls all the data users store at the server as well as all the messages between the server and the client. The adversary can launch both passive and active attacks on the application's users. Furthermore, we also assume that the cloud service itself can act maliciously. In addition to that, we assume that our browser extension along with the client's browser and host is not compromised.

IV. Securing Google Docs (SecGOD)

In this section, we present our protocol/tool (SecGOD) in which we make use of well known cryptographic techniques in order to preserve the privacy of the content of documents that have been created through online office suites such as google docs. Our tool is implemented as a Greasemonkey script [12] which means that can successfully run in the four most popular browsers: Firefox, Chrome, Internet Explorer and Safari. At the time that this paper is written, SecGOD is tested only with google docs, which as far as we concerned, is the most popular online office suite. Furthermore, SecGOD does not require any change to the code that is running on the server and it is implemented in such a way that all the provided functionality runs in the client side. In addition to that, SecGOD does not make any use of the API that is provided by google thus makes it an ideal solution also for applications that do not offer any API for the developers. Figure 1 illustrates a general overview of the presented protocol.

The rest of the section is divided into the following three parts:

- Installation
- Encryption/Decryption of the contents of a document
- Sharing a document with other users of the community (e.g users that they use google docs)

A. Installation

A user that wants to use SecGOD needs to first install Greasemonkey, which is a free add-on, on its browser. Once Greasemonkey is installed, managing the extension is quite user friendly.. The user can install secGod online, or can run the secGod javascript file and a Greasemonkey alert box will automatically appear asking for a permission to install SecGOD or not. After user finishes the installation procedure, every time that Greasemonkey is active and user visits google docs, SecGOD will automatically run. As you may realized, during SecGOD installation, we did not create any secret key for the user. This is because we want to ensure that no sensitive information will be saved in user's computer, even if this information can be in an encrypted form (e.g encrypted cookie). With this way, not only we protect user from local vulnerabilities such as back-doors and trojan horses but we also make SecGOD easy to be used from any computer that the user has access to.

B. Encryption/Decryption

Lets suppose that user u_i visits google docs and creates a new document d_i. Google directly sends u_i to a screen that by default contains a word-processor and an empty document. Since, u_i has already installed SecGOD, this screen will be different. More precisely, a new toolbar will be available at the very top of the page, which is inserted dynamically to the

document object model by the SecGOD javascript code. This toolbar contains the following:

- A new editor where u_i will be able to edit d_i
- A field where u_i is prompted to enter a password (secret key) with which the encrypted text $E(d_i)$ of d_i will be calculated
- The key size of the algorithm (128 bits, 192 bits, 256 bits)
- An encrypt button that will calculate $E(d_i)$

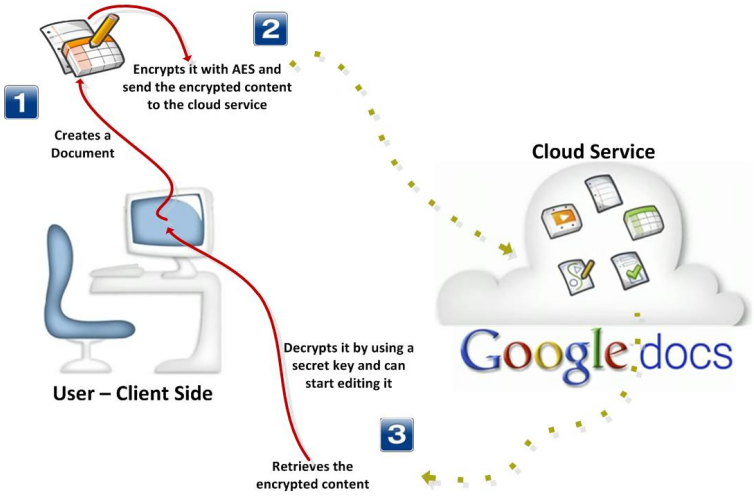

Fig. 1. General Overview of SecGOD

For the encryption/decryption process we use the symmetric key algorithm[1] *Advanced Encryption Standard (AES)* [13] and the procedure is as follows. When u_i presses the encrypt button the contents that exists in the SecGOD editor are encrypted with AES. As a key, we use $\mathcal{H}(pass)$ where $\mathcal{H}$ is a cryptographic hash function and $pass$ is the secret key that u_i added to the password field. After $E(d_i)$ is calculated, the encrypted content is sent via dispatched Javascript keypress events, that our script generates, to the default editor of google docs[2]. This means that, SecGOD does not implement the save procedure but google, which is responsible for saving the encrypted content to its cloud servers. From now on, u_i can see at the same time the original content of d_i as well as the encrypted one $E(d_i)$. At this point, we have to mention that the encrypted text which is transfered to google docs editor is not editable. With this way, we can avoid possible mistakes (e.g accidentally insert text into the encrypted one) from user, which could lead to "destroy" the ciphertext and therefore the decryption of $E(d_i)$ will result to a nonsense text.

Now that u_i has created an encrypted document will have to be able to decrypt it every time that needs to edit it. So, when u_i opens d_i, first she sees the encrypted text in the editor provided by google. Then she must press the decrypt button and give the password to an alert box that opens. On submission, $\mathcal{H}(pass)$ is calculated and automatically is being used as the key to decrypt $E(u_i)$. Then, the decrypted text is loaded in SecGOD editor and u_i can edit it.

C. Secure Sharing

One of the most important functionality that google docs and online office suites in general offers, is the ability to share documents between users. Multi-user collaboration though adds some complications since a secret key must be securely and efficiently shared between the users. As before, lets assume that u_i creates a document d_i that she also wants to share with k users from U. At this time, the only user who knows how to decrypt d_i is the creator of the document (u_i) and is the one who has to share this secret information with the users from U_{d_i}[3]. This problem could be easily solved with the use of asymmetric-key cryptography. For example, each user (u_j) would have a public/private key pair (k_{u_j}/K_{u_j}). The public key would be known to each member of the community, while the secret key would be kept secret. If u_i wanted to share the password in a secure way with all u_j in U_{d_i} she would simple encrypt the password with the public key of each user in U_{d_i} and would send to each one the following $\{pass\}_{k_{u_j}}$. Upon reception, u_j would decrypt the message with K_{u_j} and would find the password needed to decrypt the document. Even though this solution would solve the problem, it would either require to change the core functionality of the code that is running on the server or it would need to implement a whole web application that would make use of google docs API while at the same time we would have to build a community based on SecGOD's users. Both ways are considered as inefficient so we propose a technique which we admit it is not the best way to share a secret, but in comparison with the existing solutions [7], [8], [10] it is an improvement towards this direction.

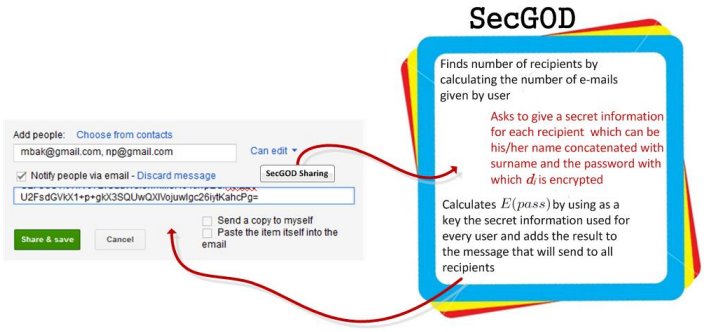

Fig. 2. Securely sharing the secret between participants

Our solution for sharing the password ($pass$) with which d_i is encrypted, is based on the fact that u_i will know at least some basic information about each $u_j \in U_{d_i}$. This information

[1]In a symmetric key algorithm, the same key is used for both encrypting and decrypting the data.

[2]We send text as key presses to google docs through the code: *document.getElementsByTagName("iframe")[0]. contentDocument.dispatchEvent(keyEvent);*

[3]U_{d_i} is a subset of U and contains all the k users with which u_i needs to share the document d_i. U_{d_i} has already been defined in Section III.

can be anything, like name and surname, cell phone number, birth date, name of the company that u_j works for etc. So, when u_i requests to share d_i, she will first have to enter the addresses of the people that she wants to share the document with. Next, she has to select the option "Notify people via email" and automatically a *SecGOD* button will appear. When u_i presses this button SecGOD first asks her to give the password for d_i which is temporarily stored into the client. Second, SecGOD finds the number of recipients (based on the e-mail addresses that u_i have added) and for each recipient (u_j) asks u_i, via a pop-up window, to write an information s_{u_j} (such as name, surname etc) about u_j. Then, SecGOD encrypts $pass$ with s_{u_j} and adds the result ($E(s_{u_j})$) to the e-mail message as shown in figure 2. When u_i presses the "Share & save" button, the password that it is stored on the client is immediately deleted and each u_j will receive a message that will contain the encrypted password.

Upon reception, each u_j can connect to google docs and open d_i. At this point, u_j can only see the encrypted content without being able to edit it. In order to decrypt d_i and be able to edit it, u_j presses the "Retrieve Password" button and gives as input $E(s_{u_j})$ as well as the secret s_{u_j} with which the password is encrypted. SecGOD is decrypting $E(s_{u_j})$ and reveals $pass$ to u_j. So, now u_j owns the password with which she can decrypt and start editing or simply reading d_i.

V. Security Analysis

In this section we analyze the behavior of SecGOD in the presence of adversaries. First, we consider the case of a malicious cloud service provider (CSP) and then we discuss the case of an attacker that colludes with CSP. Attacks that assumes that the client side code is infected with malicious software is outside the scope of this paper.

As we described in Section IV all the operations of SecGOD are taking place on client's machine through the use of Javascript. This means that when a user u_i saves a new document d_i, before the contents are sent to CSP, they are encrypted with a key that the user generates. We assume that the u_i always keep this key secret. After the encryption, CSP will only receive the ciphertext ($E(d_i)$) of the the document. So, CSP can only break the security of SecGOD by only implementing a ciphertext-attack on $E(d_i)$.

One of the most popular block ciphers is represented by AES. AES comes in three versions (AES-128, AES-192, and AES-256), for which the last 3 digits of the name represent the size of the key (e.g. AES-128 requires a 128 bit key size). AES has been the target of a number of hacking attempts. For example, regarding AES-128, there is no known attack which is faster than the 2^{128} complexity of exhaustive search. The time required to break the other two versions is significantly different for AES-192, an attack with a duration of 2^{176} is required, while for AES-256 a duration of 2^{119} is necessary. Although the durations of these attacks are significantly lower than the duration of an exhaustive search, from practicality reasons they are considered to not be major threats for AES-based systems [14].

There are though works [14], [15], [16], [17], [18], that shows that AES can be broken with a low complexity. However these attacks are based on the unrealistic model of related-key attacks [19] where the attacker can observe the operation of a cipher under several different keys whose values are initially unknown, but where some mathematical relationship connecting the keys is known to the attacker. For example, the attacker might know that the last 80 bits of the keys are always the same, even though he doesn't know, at first, what the bits are.

Regarding the case where an attacker colludes with CSP, the security of our protocol is again based on the fact that breaking the encryption of AES is considered a hard problem. However, since the attacker can collude with CSP we assume that he will eventually gain access to all the documents that user u_i has access. Nevertheless, the attacker cannot violate the privacy of the user, since he will not be able to break the encryption. The only damage that he can do is to either make changes to chiphertext, which means that the user can no longer correctly decrypt the documents, or to completely remove them from the cloud service.

VI. Experimental Results

This section, presents the implementation of SecGOD. In order to prove the effectiveness of SecGOD, we implemented our protocol as a Greasemonkey script. For the encryption and decryption process, we used AES 128, AES 192 and AES 256. In addition to that, we used five different texts of 799, 1437, 6945, 25980, 76526 characters long respectively and we run the encryption and decryption algorithm 1000 times for each text and each algorithm respectively. Then, for each encryption and decryption process we calculated the average time for encryption and the average time for decryption.

Figures 3, 4 and 5 displays the results following 1000 test runs in a laptop computer with a 2.50GHz CPU and 8GB RAM, running on Windows 7. As we can see from the provided figures, AES 128 is the fastest one while AES 256 needs more time in order to complete the cryptographic operations. More precisely, in the case where we add to SecGOD a small text of 799 characters we have the following results:

- **AES 128:**
  - Average Time for Encryption: $16ms$
  - Average Time for Dencryption: $17ms$
- **AES 192:**
  - Average Time for Encryption: $19ms$
  - Average Time for Dencryption: $18ms$
- **AES 256:**
  - Average Time for Encryption: $23ms$
  - Average Time for Dencryption: $23ms$

In the case where we add to SecGOD a big text of 76526 characters we measured the following results:

- **AES 128:**
  - Average Time for Encryption: $143ms$
  - Average Time for Dencryption: $156ms$

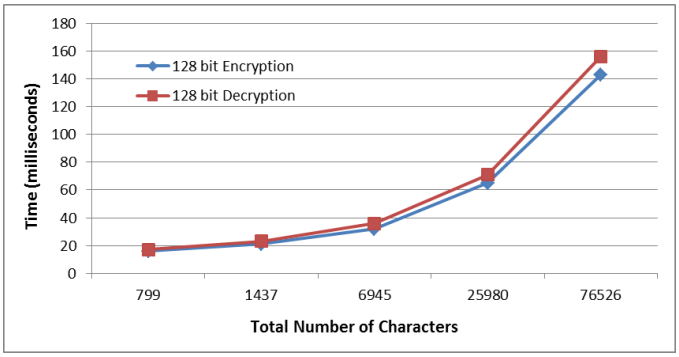

Fig. 3. 128bits Encryption/Decryption

- **AES 192:**
  - Average Time for Encryption: $182ms$
  - Average Time for Dencryption: $197ms$
- **AES 256:**
  - Average Time for Encryption: $201ms$
  - Average Time for Dencryption: $213ms$

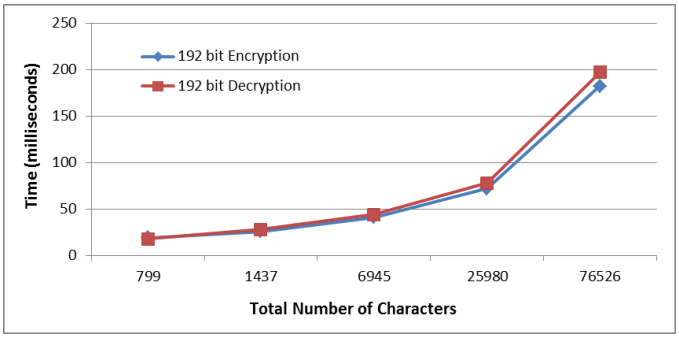

Fig. 4. 192bits Encryption/Decryption

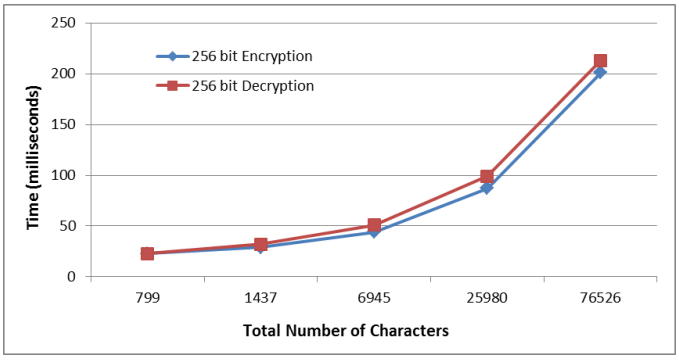

Fig. 5. 256bits Encryption/Decryption

VII. Conclusions

In this work we presented SecGOD, a protocol for securing the contents of documents in online office suites. Our protocol relied on the use of symmetric key encryption and is implemented as an extension that can be successfully run through Firefox, Chrome, Safari and Internet Explorer. In addition to that, SecGOD is implemented in Javascript and all the operations take part at the client side. This means that no changes need to be done at the server side code that is provided through the cloud service provider. The effectiveness of SecGOD is demonstrated by conducting extensive experiments measuring the processing time for the three versions of AES (128, 192, 256 bits).

A. Future Work

As part of our future research, we intend to enhance the functionality of SecGOD by developing a different kind of encryption procedure. More precisely, we are planning to divide the document into blocks and apply the encryption/decryption process to each block rather than the whole document. In addition to that, we will test the performance of SecGOD if we automatically, with the use of Ajax, update the encrypted text as the user edits the original text (no need for pressing the encrypt button). Finally, as SecGOD is implemented now, users are not able to use some core features, such as spell checking and translation, that most of the online office suites provides. We plan to conduct research on finding ways to integrate most of this functionality to our protocol. Furthermore, a very challenging problem is to enhance our protocol in such a way that importing an image and drawing a picture will be able and secure as well. This means that when for example a user imports an image in the document, this image must be also encrypted and then saved to the cloud service provider servers. Finally, what we see as a good solution for making the document sharing and as a consequence the multi-user collaboration more secure, is to create a web based service that will make use of google API and each user will be able to create groups of people. The novelty will be in the fact that each group will share a unique public/private key pair with which all the documents that are shared between the users of this particular group will be encrypted.

References

[1] CNN, "Dropbox's password nightmare highlights cloud risks," 2011.
[2] CNN, "More than 6 million linkedin passwords stolen," 2012.
[3] D. C. Plummer and P. Middleton, "Predicts 2012: Four forces combine to transform the it landscape," tech. rep., Gartner.
[4] M. Christodorescu, "Private use of untrusted web servers via opportunistic encryption," in *In Web 2.0 Security and Privacy*, 2008.
[5] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell, "Stronger password authentication using browser extensions," in *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, SSYM'05, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2005.
[6] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, pp. 792–807, Aug. 1986.
[7] F. G.-C. Lilian Adkinson-Orellana, Daniel A. Rodrguez-Silva and J. C. Burguillo-Rial, "Privacy for google docs: Implementing a transparent encryption layer," in *2nd Cloud Computing International Conference (CloudViews 2010)*, (O Porto, Portugal), May 2010.
[8] G. D'Angelo, F. Vitali, and S. Zacchiroli, "Content cloaking: Preserving privacy with google docs and other web applications," in *PROCEEDINGS OF 25TH ACM SYMPOSIUM ON APPLIED COMPUTING (SAC) 2010, WEB TECHNOLOGIES TECHNICAL TRACK*, ACM, 2010.
[9] "gdocsbar." www.gdocsbar.com.
[10] Y. Huang and D. Evans, "Private editing using untrusted cloud services," in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pp. 263 –272, june 2011.

[11] Wikipedia, "Online office suite — wikipedia, the free encyclopedia," 2012. [Online; accessed 21-June-2012].
[12] Wikipedia, "Greasemonkey — wikipedia, the free encyclopedia," 2012. [Online; accessed 28-June-2012].
[13] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Berlin, Heidelberg, New York: Springer Verlag, 2002.
[14] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, "Key recovery attacks of practical complexity on aes-256 variants with up to 10 rounds," in *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, (Berlin, Heidelberg), pp. 299–319, Springer-Verlag, 2010.
[15] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full aes-192 and aes-256," in *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, (Berlin, Heidelberg), pp. 1–18, Springer-Verlag, 2009.
[16] A. Biryukov, D. Khovratovich, and I. Nikolić, "Distinguisher and related-key attack on the full aes-256," in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, (Berlin, Heidelberg), pp. 231–249, Springer-Verlag, 2009.
[17] J. Kim, S. Hong, and B. Preneel, "Related-key rectangle attacks on reduced aes192 and aes-256," in *Proceedings of Fast Software Encryption (FSE 07), Lecture Notes in Computer Science*, pp. 225–241, Springer-Verlag, 2007.
[18] W. Zhang, L. Zhang, W. Wu, and D. Feng, "Related-key differential-linear attacks on reduced aes-192," in *Proceedings of the cryptology 8th international conference on Progress in cryptology*, INDOCRYPT'07, (Berlin, Heidelberg), pp. 73–85, Springer-Verlag, 2007.
[19] E. Biham, "New types of cryptanalytic attacks using related keys," in *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, EUROCRYPT '93, (Secaucus, NJ, USA), pp. 398–409, Springer-Verlag New York, Inc., 1994.