MEG: The Email Mobile Encryption Gateway

Gregory Rehm, Michael Thompson, Brad Busenius, and Jennifer Fowler

University of California, Davis, Argonne National Laboratory
grehm@ucdavis.edu {thompsonm,bbusenius,jfowler}@anl.gov

May 23, 2016

Abstract

Email cryptography has long been considered a problem that was too hard to practically solve. MEG, or the Mobile Encryption Gateway aims to fix the quandaries associated with email encryption by ensuring it is easy to perform all while maintaining data security. MEG performs automatic decryption and encryption of all emails, allowing users to not worry about the internal workings of how encryption is performed. MEG is meant to be email client agnostic enabling users to use any mail service they desire to send messages. Encryption actions are performed on the users mobile device meaning keys and decrypted data stay personal to the user. Most importantly, MEG is end-to-end encrypted ensuring that a users information stays private only to them. As a result we hope that MEG will finally provide supply the solution to the problem of practically encrypting email.

1 Introduction

Cryptographically secure email has seen many approaches attempting to make it widely available. Yet, for varied reasons all attempts have either failed or have not reached widespread adoption. The most common issue reported is users have difficulty using encryption technologies without specialized training. And even with special training most average users still fail at encrypting their email [19]. Other detractions from the mass usability of email encryption include the inability to verify the identity of contacts, a lack of end to end encryption, network effects, and payment being required for services rendered.

To solve this problem we propose MEG as a secure, free, and highly usable alternative to all other previously devised technologies. First we show why previous attempts to encrypt email en masse have failed. We then show how MEG aims to address all these deficiencies. Next we discuss that the MEG architecture enables us to create a secure, end to end encrypted system that is email client agnostic; enabling users to keep their email provider by installing an additional plugin for their browser or mail application. Finally we display MEGs usability characteristics and how it enables users to perform all necessary actions to securely encrypt messages without forcing users to understand encryption itself.

2 Background

Underneath the majority of the communications performed on the Internet today is encryption. Encryption ensures that our bank records remain private to snooping eyes and that our store purchases stay secret. The basis for one of the most common types of encryption technology on the Internet dates back to 1976 when Diffie and Hellman proposed public key cryptography. Since then encryption has slowly increased in relevance to electronic communications. However after the 2013 Snowden revelations major corporations and average computer users alike were shown just how vulnerable their data is to prying eyes. Since then there has been a flurry of activity towards encrypting all communications over the internet [6]; and yet three years later, despite the rise in transit encryption using TLS by providers like Google [11], most email traffic remains unencrypted end to end. The reasons for this are complex but often boil down to a lack of usability of email encryption and network effects. Quite simply the average user neither has the time, money, willingness, or technical knowledge to fully encrypt their email especially when free and intuitive services like Gmail and Yahoo Mail abound [9]. This in turn creates a feedback loop where network effects kick in. Because few people already encrypt their emails others are less likely to encrypt their communications because if they did

their contacts would not be able to read them [4].

2.1 Email Encryption: What is Required

Throughout the years email encryption has seen dozens of applications developed in the attempt to enable widespread email encryption. All of these applications do the following four things

- *Sign* the message with the senders private key. This validates the identity of the sender
- *Encrypt* the message and ensure only the desired recipients can read it.
- *Verify* the original signature from the sender.
- *Decrypt* the message when received by the recipient so the message can be read.

Certain encryption methods perform some of these tasks differently. PGP works where anyone with a PGP key can sign anyone elses key. This validates identity through a concept called *web of trust*. In contrast X.509 certificates are only signed by a Certificate Authority (CA) which are implicitly trusted by name.

2.2 S/MIME

The reigning standard of email encryption is S/MIME. S/MIME has the ability to perform end to end encryption for email and is backed by CAs that are able to validate the indentity of those we are communicating with. The problem with S/MIME is the issue of certificate creation. Creation of X.509 certificates is a centralized process imposed by the CAs [10]. The CAs charge expense for their services, anathema to many users used to free services. In addition the process of obtaining a certificate can take days if not weeks. Even if a user is willing to go through the trouble of obtaining a certificate network effects must be taken into account. If a mail recipient does not have an S/MIME certificate themselves then email cannot be encrypted. There exist email services like Hushmail that offer to automate this process but payment is required for these services to offset cost of obtaining certificates from the CAs. Often third party paid services do not make their code is audit-able and we cannot ensure that it is end to end encrypted. Corporations often provide the service of providing S/MIME certificates for their employees as well but this only ensures encryption will occur between members of the same corporation and not necessarily with parties outside the corporate boundaries. As a result we cannot rely on corporations or paid services to fix S/MIMEs deficiencies.

The best attempt yet to fix the deficiencies with S/MIME is named Key Continuity Management (KCM) by Simson Garfinkel and offered to provide automated generation of self signing S/MIME certificates [10]. Theoretically this would mean anyone could generate an S/MIME certificate for free and instantaneously. In this manner, network effects could slowly be resolved over time. However many mail providers do not accept self signed certificates because they are a security vulnerability. Their main problem is that you cannot verify the identity of a person with a self signed certificate. Garfinkel later tried to argue that if you could let users determine whether to trust the host the message is coming from similar to the way SSH works [9]. However this does not constitute a feasible solution to the problem. Asking the user to understand their security risk is unreasonable. The vast majority of users do not understand the purpose of self signed certificates [5]. When study participants were prompted with security decisions in a study on HTTPS on whether or not to accept dubious certificates users frequently chose to accept them [1]. In fact, according to Downs a likely way to spoof identity would just be to impersonate a company the user does business with [5]. It is because of this that major mail clients do not even accept the use of self signed certificates, viewing them as insecure [15]. Furthermore KCM could place users at greater security risk for attacks like phishing given that they trade provider built-in security filtering for encrypted mail.

Besides KCM there are no other feasible alternatives to using S/MIME as a mass email encryption solution. Thus due to usability headaches present in basic S/MIME and the insecurity of KCM, S/MIME cannot be considered a viable method to perform mass email encryption.

2.3 PGP

The main alternative to S/MIME is to use PGP. In contrast to generating X.509 certificates, creating PGP keys is decentralized and can be performed for free in a matter of seconds. PGP is able to validate identity through a concept named *web of trust*. In practical terms *web of trust* works through key signing. If a friend of yours that you trust has signed the key of another person, then *web of trust* would state that you should trust the identity of that other person [20]. Assuming that a user has properly performed their web of trust PGP is just as secure as S/MIME and is able to validate the identity of the

participants in a communication [8].

Despite advantages of being decentralized and free, PGP is notoriously difficult to use. In a famous study name *Why Johnny Can't Encrypt* researchers found that most study participants couldn't even encrypt their emails using GUI tools for simplifying PGP actions. The few that were able to encrypt their communications were prone to displaying their private key in plaintext; defeating the entire purpose of encryption [19]. Followup studies with improved PGP GUI tools have had no more success with study participants than the original trial [17]. As a result we don't believe that further advances in PGP GUI tools will improve PGP's usability headaches. Once again there exist paid services to automate PGP but they too suffer from lack of end to end encryption and have unauditable code [2, 7, 13]. Most importantly these services charge money meaning widespread adoption will not occur. There does exist a free, open source email encryption application using PGP named OpenKeychain. It is superb for usability and accessibility but it requires a user to use a special email client on Android only and disallows use of clients that the user has grown accustomed to [16].

2.4 MEG

MEG aims to address all the problems that stem from using PGP, S/MIME, and mail provider based encryption schemes. First and foremost MEG aims to use PGP because it is free, decentralized, and can validate contact identity using web of trust. MEG however aims to alleviate the usability troubles surrounding PGP by automating the entire process of key generation, signing keys, encryption, and decryption of messages for the user. This way the user gets to enjoy encryption for their emails while not having to worry about the low level details of how it works; eliminating the problems encountered in *Why Johnny Can't Encrypt* [19].

MEG will ensure user data stays private. End to end encryption will be built into MEG on any part where it is necessary. A users private key will stay on their phone. This way no one except the owner of the phone will have the ability to decrypt messages. The actual encryption of emails will be provided as a service by a mobile phone with the MEG Android app on a mobile device serving as the email gateway. MEG will then relay the email over a secure channel back to the user's mail client which can forward the email to its intended destination.

Finally to accomplish the task of combating network effects within encryption MEG aims to provide email based invitations to anyone without the service. This operates in similar fashion to how social networks like Facebook grew in popularity, through electronic word of mouth [18].

3 Architecture

The architecture of the MEG system is what enables it to be both secure, private, and flexible enough to ensure people can still use their existing mail clients. The entire system consists of three components: A mobile app that performs all PGP related actions like key generation, encryption, decryption, and signing. A mail client plugin that acts to send messages to the phone for encryption or decryption before they are either sent to a recipient or read. Finally, a server component acts as a PGP public keystore, storage for revocation keys, and a message broker between the client and android device. As of this moment only a Thunderbird email plugin and an Android app have been completed for MEG.

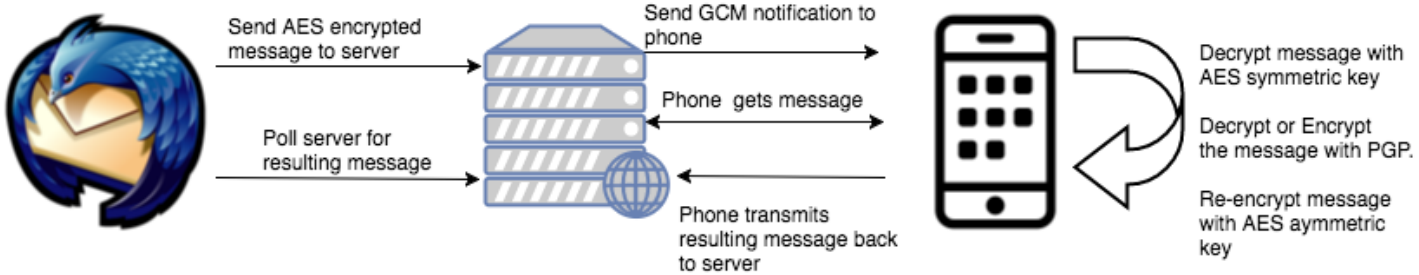


Figure 1: MEG: Message Flow

Upon installing the Android app the user is directed to the installation page that gets a sufficient amount of information from the user to generate the PGP public/private key pair and a revocation key. The revocation key is meant to be used if the user loses their phone or has their password compromised. Activating this key will then ensure that no one else will trust their public private key pair in the future.

To ensure that communications between the mobile app and the mail client plugin remain secret we ensure that all communications between these two components are encrypted via an AES-256 symmetric key. Because of the AES encryption even if an attacker gained access to the MEG server component then electronic communications would not be able to be read. This ensures that MEG is end to end encrypted.

Because mobile devices are frequently networked behind routing devices or ISP controlled networks that block incoming communications we must use the server component to assist in routing messages. In consequence, we engineered the mobile device to request the server for messages that are awaiting processing. To ensure this does not cause delay is email processing we send a Google Cloud Messaging (GCM) notification to the phone first to alert it that a message is awaiting processing. Upon receiving this notification the phone will then retrieve the message and perform whatever PGP action it needs to. After processing the phone will send the message back to the server for the client app to retrieve. The complete flow of how a message is transported from the client plugin is illustrated in Figure 1.

Since all encryption actions are performed on the phone email data stays private. Private keys are located on the phone as well. The private key is already protected by a password and the proliferation of data encryption on the phone ensures that the users private key remains doubly secure. In case of loss of phone or if the key password is compromised the user can revoke a key by making a request to the server. This will cause an email to be sent by MEG to the users email account for validating that they actually made the request. This way we can ensure that malicious actors cannot revoke MEG PGP keys.

The mail client plugin is engineered to guarantee that MEG stores no plaintext data for prying eyes to view. Email is stored on the server in encrypted form so the mail provider will have no access to its contents. In order to decrypt a message the MEG plugin looks for a special header attached to all MEG emails and then transmits them to the phone for decryption. When the messages arrive back on the client they are decrypted with the AES-256 symmetric key and then displayed on screen through Javascript injection. However this display is ephemeral and only lasts as long as the user has the message window open. When the user closes the message no decrypted information is stored in a database and the decrypted information is garbage collected. In Figure 2 we can see how the user would initially view their email at rest in the client. In Figure 3 we can see that exact same message after the Javascript injection occurs.

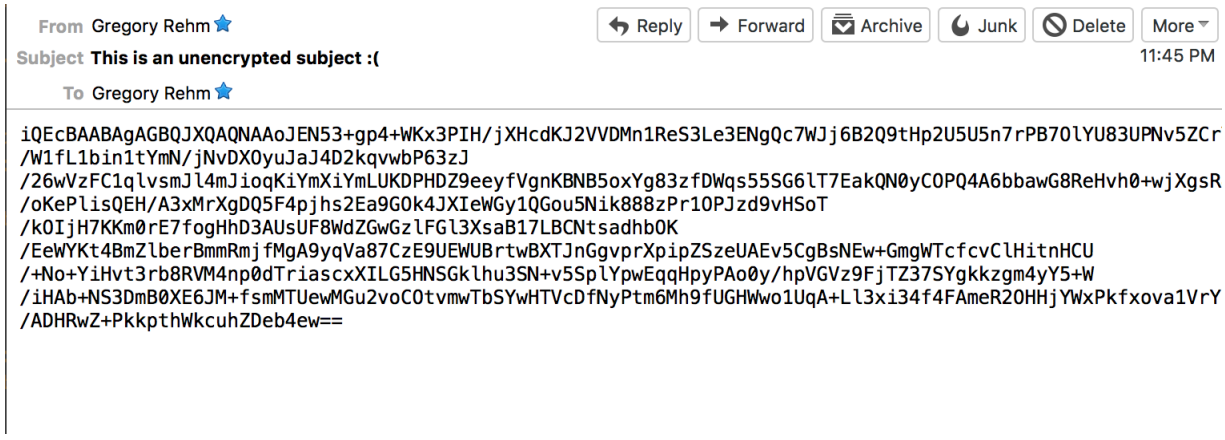


Figure 2: An encrypted message in the mail client

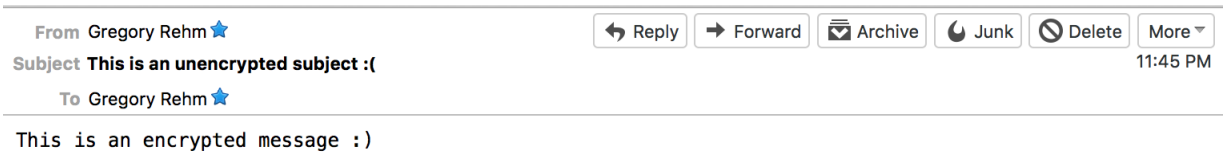
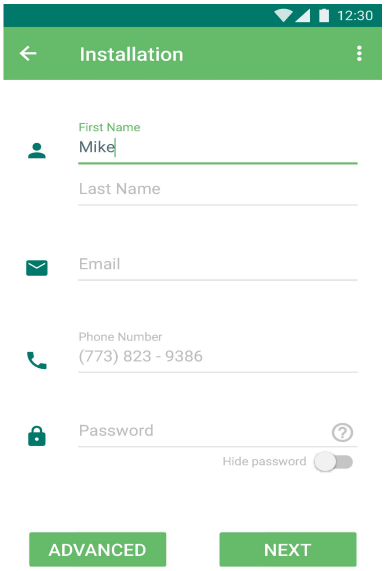


Figure 3: The same message but decrypted after Javascript injection

4 Usability

Our primary aim was to make MEG as usable as possible to ensure people with little technical training would be able to encrypt their mail. In the MEG system there are two components that users are meant to interact with: the mail client plugin and the mobile app. First we will show how the mobile app's minimal UI makes generating PGP keys as painless as possible for users. We also show that the generation of symmetric key information is does not require any technical skill. Then we show how the mail client plugin enables users to use the same mail clients they are familiar with to send encrypted mail just by the click of a button.

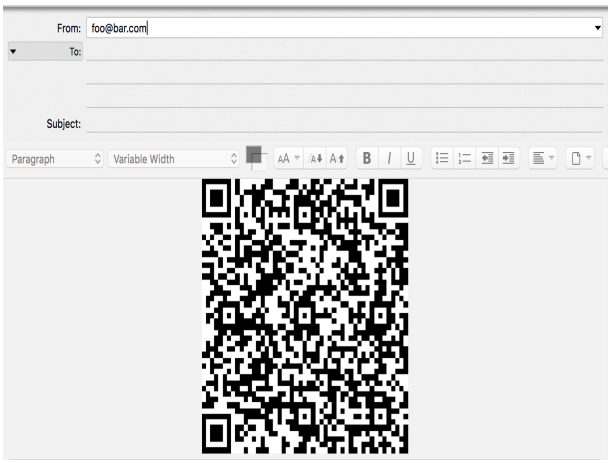


4.1 Mobile App Usability

The MEG mobile app is where all PGP encryption, decryption, signing, and key generation actions take place. Fortunately, users need to perform no manual work on their ends to accomplish this task themselves. The app is also designed to be as minimal as possible. First of all, are only four different UI panels that the user can actually navigate to ensuring that we reduce navigation confusion. Second, the app is designed to make the process of generating a PGP key as simple as possible. Below we see that only five pieces of information are needed from the user to generate a PGP public/private key pair: first and last name, phone number, email, and password. After inputting this information the user is instructed to proceed after which a PGP public/private key pair will be generated for them so they can start using MEG.

After generating a public private key pair the MEG app needs to scan a QR code to authenticate a mail client plugin. In security applications QR codes have shown interesting promise. QR codes have even been used for physical access control [14] and can securely transmit PGP information [3]. In MEG, we use a QR code to transmit AES-256 symmetric key information from the mail client plugin to the mobile app. This symmetric key information is then used to encrypt communications between the mail plugin and the the phone. The symmetric key is generated in the mail client plugin as shown below. After the QR code is scanned the image is removed from the screen ensuring that malicious actors cannot gain access to the symmetric key information. The main advantages of using a QR code to to transmit symmetric key information is that it is both secure and easy to use. Most users are well acquainted with using their mobile devices camera. Furthermore the use of a QR code precludes the necessity of performing

additional password input on the mail client ensuring that the user only needs to remember one password when using MEG.



A negative of some mail encryption applications is that a user must continually input their password each time they want to send and receive email. This is especially true of server based applications because caching a decrypted private key would mean the service is not end to end encrypted and that the application could access the users' encrypted files. Because all encryption actions occur on a users phone, and the phone is in physical possession of a user, MEG is able to avoid the user having to re-enter their password each time an email is encrypted or decrypted. Instead, MEG implements private key caching. Upon starting MEG mobile app users will need to enter their password to unlock their private key and then the key is cached and is able to be reused as long as the app stays open. This ensures that users can encrypt and decrypt their mail merely by having their phone on and MEG open. If the mobile app is closed by the user or the OS for some reason then the user will need to log back into the mobile app. Ultimately this will be detected if a user attempts to either send or decrypt an email on the client. The client will then alert the user they need to log back into the mobile app to complete their action.

4.2 Client Usability

While the mobile app performs all PGP actions, the client plugin is where the majority of user interaction will occur in MEG. As such we designed it so that users would have to do as few actions as possible to encrypt their messages.

The two main actions a user will need to perform on the mail client are generating a QR code so symmetric key data can be sent to the phone, and sending encrypted mail. The QR code is generated on the first attempt the user makes to send a piece of encrypted mail. Once scanned the QR code is removed from screen through user confirmation the code was scanned successfully. After confirmation users can begin to send encrypted mail.

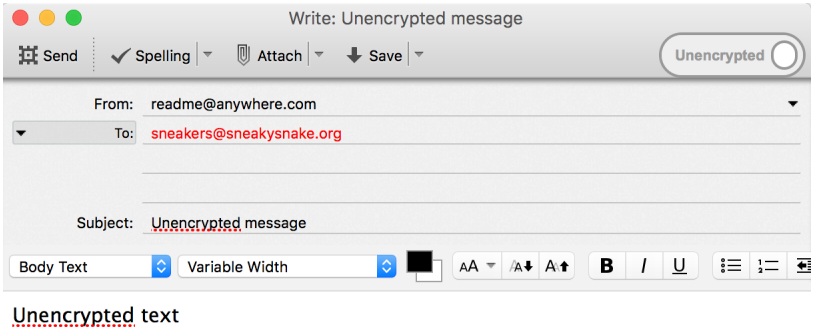


Figure 4: Client UI for sending unencrypted messages

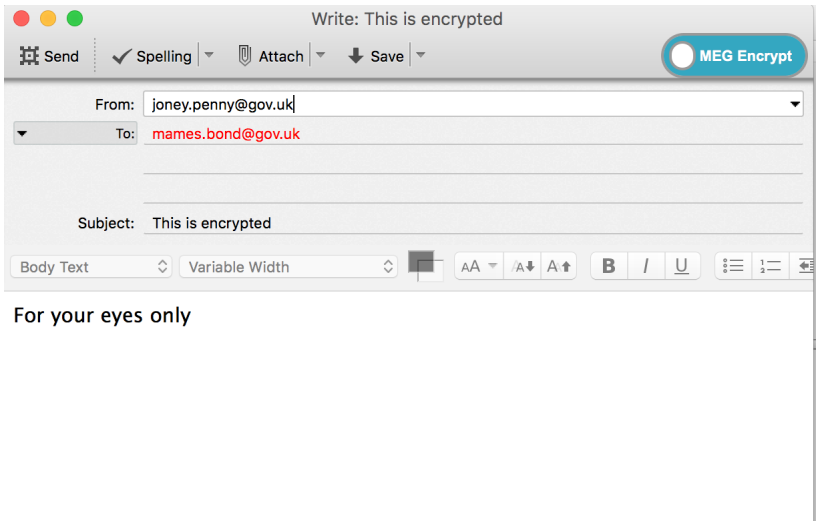


Figure 5: Client UI for sending decrypted messages

Encrypted mail is toggled through a checkbox at the top of the mail client. We always give the user the option to have their mail unencrypted in case some one in their network is unable to read their encrypted mail. This is shown in Figure 4 Encrypting mail is as easy as just clicking the encrypt checkbox and then we will be able to cryptographically secure our mail. We show its implementation in Thunderbird in Figure 5.

Once the user is finished composing their mail they only need to click the send button. As a result learning a new flow for sending an encrypted email is not necessary beyond clicking a checkbox. This is a hurdle that Thunderbird by default and some mail plugins like Enigmail introduce to their users [12]. MEG is able to avoid this problem mainly because PGP is handled automatically on Android but also because of its overall focus on usability.

5 Future Work

We have shown here how MEG offers a secure and accessible application to perform email encryption. Future directions should focus on making the software as accessible as possible. The requirement to scan a QR code may be onerous for users so it is possible there are other routes to disseminate symmetric key information from using the server as an origin over HTTPS. It has also been suggested that IPv6 would allow us to eliminate the server altogether as a message router allowing the client plugin to directly communicate with our phone. We'd also like to start work on an IOs app and a Gmail plugin to ensure that more users can have access to MEG.

6 Conclusion

MEG is a free, mail client agnostic, encryption application that automates the pain points of email encryption. MEG ensures that users do not have to worry about the details of securing their messages. These characteristics mean that MEG will be both more secure and more accessible than other email encryption schemes. MEG has the benefit of being able to learn from its predecessors. We have the ability to understand where previous schemes such as KCM failed. And we also have the benefit of being able to use the ubiquity of smartphones to serve as email gateways. We believe that the choice between security and usability does not have to be a major tradeoff for users. It is rather just a matter of engineering the correct solution that average people can use. We believe for email encryption that MEG is that solution.

References

[1] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security and Privacy*, 7(1):78–81, 2009.

[2] ciphermail. Email encryption gateway. `https://www.ciphermail.com/gateway.html`. Accessed: 2016-02-17.

[3] John Denker. Distributing pgp keys and fingerprints. `https://www.av8n.com/computer/htm/distributing-keys.htm`. Accessed 2016-05-21.

[4] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *WEIS*, 2006.

[5] Julie S Downs, Mandy B Holbrook, and Lorrie Faith Cranor. Decision strategies and susceptibility to phishing. In *Proceedings of the second symposium on Usable privacy and security*, pages 79–90. ACM, 2006.

[6] Klint Finley. Encrypted web traffic more than doubles after nsa revelations. `http://www.wired.com/2014/05/sandvine-report/`. Accessed: 2016-05-20.

[7] Electronic Frontier Foundation. Secure messaging scorecard. `https://www.eff.org/secure-messaging-scorecard`. Accessed: 2016-02-17.

[8] Steven M Furnell, Nathan Clarke, Cristian Thiago Moecke, and Melanie Volkamer. Usable secure email communications: criteria and evaluation of existing approaches. *Information Management & Computer Security*, 21(1):41–52, 2013.

[9] Simson L Garfinkel, David Margrave, Jeffrey I Schiller, Erik Nordlander, and Robert C Miller. How to make secure email easier to use. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 701–710. ACM, 2005.

[10] Simson L Garfinkel and Robert C Miller. Johnny 2: a user test of key continuity management with s/mime and outlook express. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 13–24. ACM, 2005.

[11] Google. Email encryption in transit. `https://www.google.com/transparencyreport/saferemail/`. Accessed: 2016-05-20.

[12] Ludwig Hugelschafer, Daniel Raffo, Patrick Brunschwig, and Robert J. Hansen. Openpgp email security for mozilla applications; the handbook v 1.8. `https://www.enigmail.net/documentation/Enigmail_Handbook_1.8_en.pdf`. Accessed 2016-05-23.

[13] Hushmail. How hushmail works. `https://www.hushmail.com/about/technology/how-it-works/`. Accessed: 2016-02-17.

[14] Yung-Wei Kao, Guo-Heng Luo, Hsien-Tang Lin, Yu-Kai Huang, and Shyan-Ming Yuan. Physical access control based on qr code. In *Cyber-enabled distributed computing and knowledge discovery (CyberC), 2011 International Conference on*, pages 285–288. IEEE, 2011.

[15] Howard Lightstone. How can i force use of self-signed ssl certificate. `https://productforums.google.com/forum/#!topic/gmail/6gODk9n65ZU`. Accessed: 2016-02-17.

[16] OpenKeychain. About. `https://www.openkeychain.org/about/`. Accessed: 2016-05-19.

[17] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, pages 3–4, 2006.

[18] Michael Trusov, Randolph E Bucklin, and Koen Pauwels. Effects of word-of-mouth versus traditional marketing: findings from an internet social networking site. *Journal of marketing*, 73(5):90–102, 2009.

[19] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Usenix Security*, volume 1999, 1999.

[20] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995.