Viable Email Attacks and a Simple End-to-End Security Solution

** Shushan Zhao*

Abstract

Many attacks, scams, and malware threats are based on or spread through emails nowadays. Although people have been fighting against them with technical and legal measures for many years, the situation has not improved. It seems to be getting worse and worse. We attribute this to lack of end-to-end security measures for emails in current internet infrastructure. Most past security solutions provide either in-domain authentication, or domain-to-domain authentication. Available end-to-end solutions are based on public key cryptography and have many limitations. In this paper, we propose a simple end-to-end solution for email security. It is based on the idea of trust chain from sender to receiver, which spans multiple domains and organizations without the requirement of a uniform platform. On the client-end, it is transparent to the user without requirement of any user operation. The solution provides end-to-end authentication and integrity for its users, which is hard to achieve or use in existing works.

Keywords: Attack, email, security, signature, verification

I. INTRODUCTION

Nowadays, we cannot live or work without emails. Pervasion of smart phones and mobile computing devices, 4G highspeed data networks and Wi-Fi hotspots make people rely more on emails. Many attacks, scams, and malware threats are based on or spread through emails nowadays. Spamming emails and phishing emails are widely existing and cause serious problems, especially in mobile communications era. Some emails trick people into giving out their credentials. For example, a few weeks ago, one of the authors received an email seemingly from the president of the university, with content saying, "Here is an important document all staff has to look at. It's about school updates activities. Everyone needs to read the important information carefully." Attached in the email is a pdf file. If you open the pdf file, it says "This is a confidential document, verify your identity first" and asks for user ID and password before reading it. This is an example of phishing email. Some emails with spoofed sender addresses are used to issue false messages. As an example, in October 2013, an email that looked like it was from Fingerprint Cards, a Swedish biometrics company, was sent to a news agency, saying that Samsung offered to purchase the company. The news spread and the stock exchange rate surged by 50%. It was later discovered that email was fake [1]. Some other emails pretending to be from trusted senders are used to spread viruses, Trojan horses, and other malware.

These emails are mostly from anonymous or impersonated/spoofed addresses. Simple Mail Transfer Protocol (SMTP) specifies the process and requirements of sending an email. The email recipient sees the email as having come from the address in the "From:" header; they may sometimes be able to find the "MAIL FROM:" address; and if they reply to the email it will go to either the address presented in the "From:" or "Reply-to:" header, but none of these addresses are required to be checked [2], [3]. Sending such emails is pretty easy. We will see some examples in a later section.

In many mail servers, the default configuration is open mail relay. An open mail relay is an SMTP server configured in such a way that it allows anyone on the internet to send an e-mail through it, not just a mail destined to or originating from known users. The rationale of this design is: At the beginning of internet and email, only a few people had access to the internet, and even fewer people had email addresses, so it was common that many people were sharing an email address. This design enabled people to send email from any place they had internet access, and the unchecked "FROM" address enabled them to claim who they were or which organization they belonged to, and to receive

Manuscript received May 25, 2018; revised June 15, 2018; accepted June 16, 2018. Date of publication July 6, 2018.
*S. Zhao is Assistant Professor with Division of Management and Education, University of Pittsburgh at Bradford,300 Campus Dr. Bradford PA, 16701, USA. (email: shushanz@pitt.edu)
DOI: 10.17010/ijcs/2018/v3/i4/131647

replies at that address. Obviously, things have changed so much ever since then. Nowadays, each email sender has her/his own address; sharing an email address or sending an email from a different place is not a decent requirement any more. On the contrary, still having this enabled is a major concern now. Having realized this fact, many internet service providers block mail sent from open relay servers. This reduces the percentage of mail senders that are open relays. However, spammers and attackers have created distributed botnets of zombie computers that contain malware with mail relaying capability.Some technical solutions have been proposed to fight against email spoofing and spamming, such as DKIM, SPF, and DMARC which we are going to review in section II. However, they just cover part of the entire route of an email, and are not end-to-end. This means that the authentication is not covered from the sender host to the receiver host, for example, when sending an email from universityA.com domain, if user A writes "FROM: userA@universityB.com", DKIM/SPF/DMARC can detect domain name spoofing; but if the sender writes "userB@universityA.com" where userB is an authentic user of universityA.com, then these solutions cannot find anything wrong with it.

End-to-end authentication is difficult to achieve. This is because there is no security infrastructure from sender's end to receiver's end. Theoretically, Public Key Infrastructure (PKI), or personal certificates, can solve this problem, for example, the ideas of PGP (Pretty Good Privacy) and GPG (GNU Privacy Guard), but so far PKI is only applied on domain level, and not on individual user level. For the latter, too much burden would be added to current internet infrastructure, as each user needs to have a certificate. This incurs usability and key management issues. The limitation of the PKI-based solutions are elaborated in section II.

To summarize, most current email systems provide only domain-to-domain security. End users can choose some plug-in components for end-to-end security only if they understand concepts of public key cryptography well. In view of potential attacks in status quo, we propose a simple seamless solution to ensure end-to-end email authentication and integrity by which the receiver can verify if the sender is authentic, and the sender can verify if the receiver is authentic, and if the email content is forged or tampered en route. The solution is transparent to end users, so there is no usability or key management issue at all.

The rest of this paper is organized as follows: section II briefly reviews related work this work is based on. Section III shows the experiments I designed to test security of currently popular email systems, and results. Section IV presents the solution from overview to details. Section V evaluates performance of the scheme, discusses and analyzes security features and their limitations. Section VII concludes the paper.

II. RELATED WORK

To fight against emails sent from spoofed addresses, a number of effective systems are now widely used to enforce email authentication. These include DKIM, SPF, and DMARC on the server side, and PGP or GPG on the client side.

A. Domain Keys Identified Mail (DKIM)

As the name suggests, DKIM identifies the domain of the`email sender. The sender's email server signs the email with its private key if it is really from its domain, and sends the signature with the email to the receiver's server to verify. The public key of sender's server is published on Domain Name System (DNS) records of the sender's domain. In this way, the receiver is able to check whether an email that is claimed to come from a specific domain was indeed signed and authorized by the owner of that domain [4]. The purpose of a DKIM-signature is not to assure message integrity. As is stated in [5], "it does not even guarantee that a message author's data, as per a signed From: field, has a real name or a valid mailbox. The parts to be signed are chosen so as to identify the message unequivocally. A valid signature just states that the message did actually flow through a box operated by the sender's domain".

B. Sender Policy Framework (SPF)

While DKIM is only concerned about domain, SPF cares a little about the sending host. It provides a mechanism to allow receiving mail exchangers to check that the sender of an incoming mail from a domain is an authentic and authorized host from the sender's domain, namely a host authorized by that domain's administrators. The list of authorized sending hosts for a domain is published in the DNS records for that domain, in the form of a specially formatted TXT record. SPF checks that the IP address of the sending server is authorized by the owner of the domain that appears in the SMTP "MAIL FROM" command [6]. This only ensures the email sender is indeed in the sender's domain. After this authentication, it is still possible for attackers to replace sender address, or tamper the email content, and the receiver has no means to detect that. This does not

prevent spoofing inside a domain, and does not prevent User A using User B's name when sending an email out of the domain.

C. Domain-based Message Authentication, Reporting, and Conformance (DMARC)

DMARC itself does not specify requirements for authentication or integrity, but provides authentication reporting to senders to improve and monitor their authentication infrastructures [7]. It just defines a policy that allows a sender's domain to indicate that its emails are protected by SPF and/or DKIM. It also tells a receiver what to do to a message if neither of those authentication methods pass, put it into Spam folder or reject the message. DMARC is a higher managing layer on top of DKIM and SPF, and it does not address the limitation of the two protocols mentioned earlier.

D. PKI-based Solutions

In the software market, we see some products attempting to provide end-to-end security, such as Thunderbird with Enigmail plugin, Mailvelope browser plugin for Gmail/Yahoo/Outlook etc. They are all based on the idea of PGP or GPG. The sender generates a pair of public key and private key, sends the public key to the receiver and keeps the private secret locally, then uses the private key to sign and encrypt a message. Correspondingly, the receiver uses the received or previous stored public key to verify the signature or decrypt the message. However, there are two major issues in these software products:

1) Usability: According to a study conducted by Ruotti et al., only one out of twenty participants were able to successfully use the public and private keys properly with Mailvelope [8]. For common people without knowledge of public and private keys, it is hard for them to complete sign/encrypt and verify/decrypt operations in the email client interface.

2) Security: There are two aspects for security issue: For one thing, the key management on the client side is a tough task to handle. For another thing, the public/private key pairs or certificates used in these products are self-generated or self-signed, and the identity of a user can be easily forged.

E. Sender-side Password Authentication:

Some email systems use a password to authenticate the sender. For example, if you use the Gmail SMTP API to send an email, you must provide the correct username and password of the sender. However, this only prevents a forged email to be sent from the Gmail server. If the attacker him/herself has an email relay, this does not prevent him/her from using a forged Gmail account as sender, and whether this is successful depends on the receiver side. We name this solution sender-side Password Authentication, and it is far from an end-to-end solution.

F. Other Works in Literature

In literature, there are also a number of works related to this topic. In [9], the authors realize that the SSL/TLS system used to encrypt server-to-server email traffic can also be used to enforce email authentication, and most of existing techniques are server-oriented and transparent to the user. They propose an SSL-based anti-spoofing application that allows a client to send trusted emails and authenticate received emails using the SSL protocol. However, it uses a self-signed certificate to exchange a secure authentication message alongside the email with a view to prevent spoofing. Self-signed certificates can only protect integrity, i.e. using the public key of the sender and signature in the email, the receiver can verify if the email is modified or not. There is no way to ensure authenticity of the sender, i.e. the sender can generate a self-signed certificate by himself/herself to claim to be somebody else. In [10], the authors demonstrate an example of email sender address spoofing by TELNET, which is from spoofed address uso@uso.uso to an authentic user at gmail.com. They propose that the receiver server obtains a sender domain name by using Auto Whois from IP address described in “Received” of the header field; and then the sender domain name obtained and domain name after @ in the mail address written in “Received:” and “From:” are compared to detect spoofing. Again, this is only domain level authentication, and not individual user level authentication. In our previous work [11], the authors see the risk of potential email attacks and propose an end-to-end solution against these attacks. The potential attacks are just theoretical ones, and not tested and proven to be viable; and the solution is not implemented. In this work, we add experimental facts of viable attacks, explanation, and analysis of our implementation.

III. EMAIL ATTACK EXPERIMENTS AND RESULTS

In Section I, we suspect possibility of different attacks available in the existence of DKIM/SPF/DMARC mechanisms. In order to test and prove our thoughts, we set up a testing environment and sent emails with forged sender addresses to some

authentic email users. The receivers were institutional email users on Google/Microsoft state-of-art email server systems.

A. Experiments Overview

The cost and knowledge requirement to set up the experiment environment is within the capability of a college student. Briefly, we installed a Postfix SMTP server on a Linux virtual machine, and configured it as an open relay to send out forged emails. Next, we connected to the SMTP server through Telnet, and created emails using "MAIL FROM:", "RCPT TO:", and "DATA" commands. The SMTP sever then forwarded it to the destination.

I tested sending forged email with either forged sender address or forged domain name or both to Google Gmail accounts, and institutional email accounts on Microsoft products, and noticed some attacks were successful. In these attacks, some email servers scanned and filtered content inside the email, so that emails with certain content were classified as "spam" and put into the Spam folder, for example, an email with a link or an attachment was more likely to be filtered. This content-filtering measure is irrelevant to our topic in this paper, and we consider an attack as successful if there exists content that can reach the inbox folder of the tested account.

B. Successful Email Attacks with a Forged Sender Username and Real Domain Name to Another Real Domain

I sent an email with a forged sender username (spoofed_account) and our real domain name (pitt.edu) to a Gmail account. The email was successfully received in Inbox of the Gmail account [Fig. 1(a)]. Another email with same username and domain name was also sent to an institutional email account in another domain, and was successfully received in receiver's Inbox that was on Microsoft Office365 Outlook server [Fig. 1(b)].

C. Successful Email Attacks with a Forged Domain Name to Another Real Domain

I sent an email with a spoofed domain name (microsoft.com) from pitt.edu domain to an institutional email account in another domain (other than pitt.edu). The email was successfully received in Inbox of the receiver's account. The receiver's side was employing Microsoft Outlook Web App [Fig. 1(c)]. The email could still be successfully received in Inbox if the sender domain name was a forged (non-existing) one.

D. Successful Email Attacks Inside the Real Domain with Forged Sender Address

We sent an email with a spoofed username (someone else's) to an institutional email account in the same domain (pitt.edu). The email was successfully received in Inbox of the receiver's account. The receiver's side was employing Microsoft Office365 Outlook [Fig 1(d)]. The email could still be successfully received in inbox if the sender username was a forged (non-existing) one.

E. Experiment Results

In the above experiments, we forged some emails and successfully sent them to the receiver's inbox. With big chance, especially for institutional email users and when the sender's name seemed familiar, the receiver would open and read the email without suspicion or hesitation. In this case, with a forged sender, the attacker can distribute false information, fool the target to open a link and install some malware or adware etc., and launch further attacks. The results show that there are security flaws or vulnerabilities in currently commonly used email systems.

IV. A NON-PKI-BASED END-TO-END SOLUTION

The basic idea of our proposed solution is chain of trust, i.e., sender's email server verifies that the sender is authentic, and signs it; the receiver's email server verifies that the sender's email server is authentic and trustable, and its signature is authentic; the receiver's email server authenticates the receiver, and the receiver trusts her/his own email server. On the basis of these conditions, the receiver ensures the sender is authentic, and the sender ensures the receiver is authentic. The logic is sound and reasonable.

The underlying cryptography is a hybrid of symmetric and asymmetric cryptographic algorithms, similar to a cluster-based structure. Inside a cluster, a host is authenticated by its cluster head using symmetric cryptography; between clusters, hosts are authenticated via cluster-heads, using asymmetric cryptography. In this way, while the communication overhead between clusters (actually domains in terms of email communication,) is minimized, the in-cluster authentication is somehow passed and transferred to other clusters.

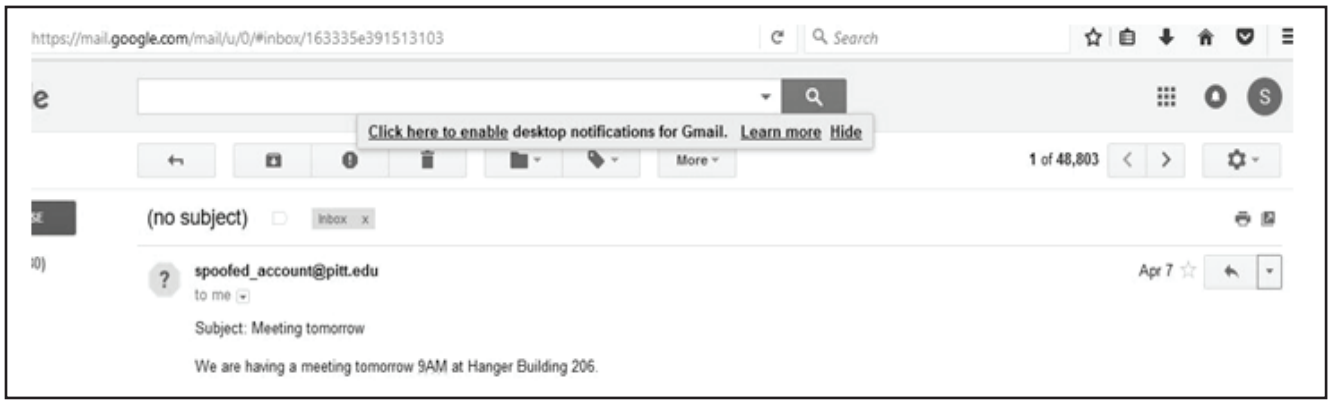

Fig. 1. Forged Emails Successfully Received in Inbox. (a) An Email With a Forged Sender Address and Real Domain Name Received by a Gmail Account

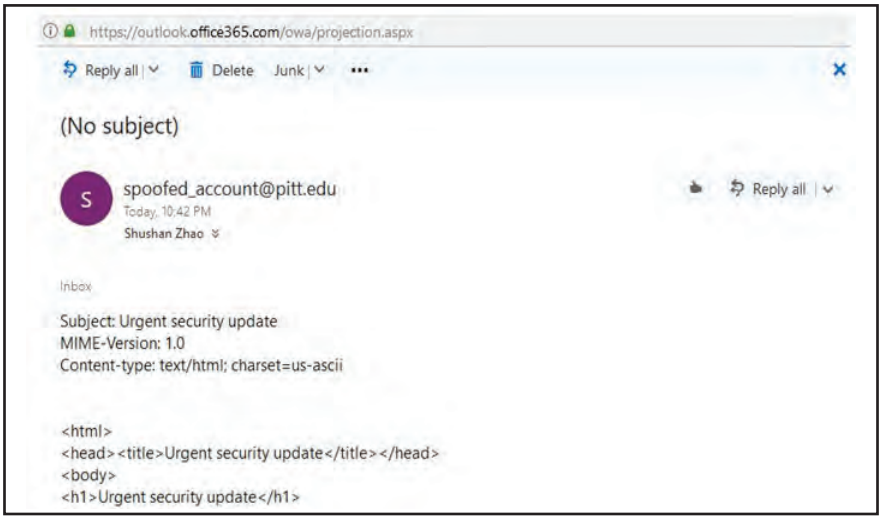

Fig. 1. Forged Emails Successfully Received in Inbox.
(b) An Email With Forged Sender Address and Real Domain Name Received by an Institutional Account in Another Domain (Employing Microsoft Office365 Outlook)

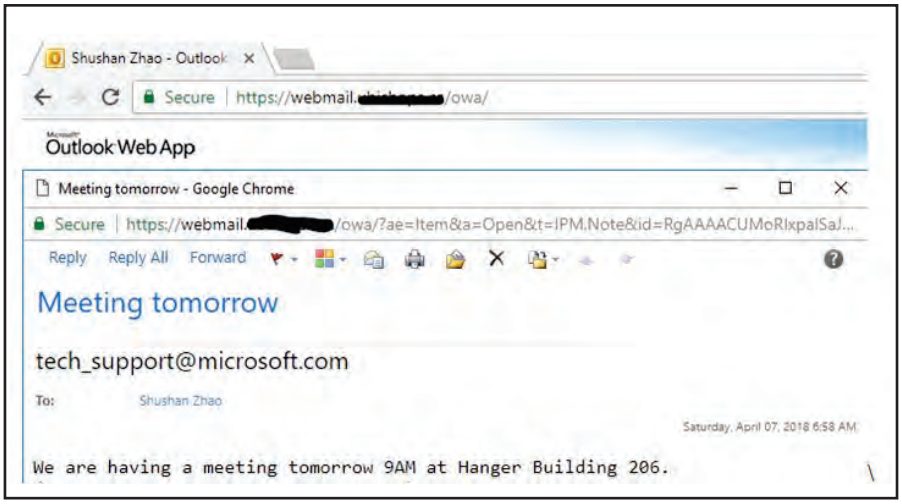

Fig. 1. Forged Emails Successfully Received in Inbox. (c) An Email With Forged Sender Address and Forged Domain Name Received by an Institutional Account in Another Domain (Employing Microsoft Outlook Web App)

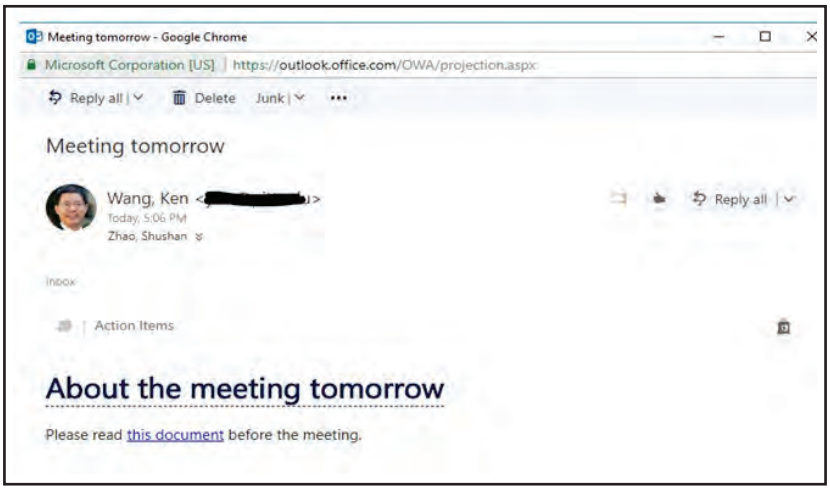

Fig. 1. Forged Emails Successfully Received in Inbox.
(d) An Email with Spoofed Sender Address Received by an Institutional Account in the Same Domain

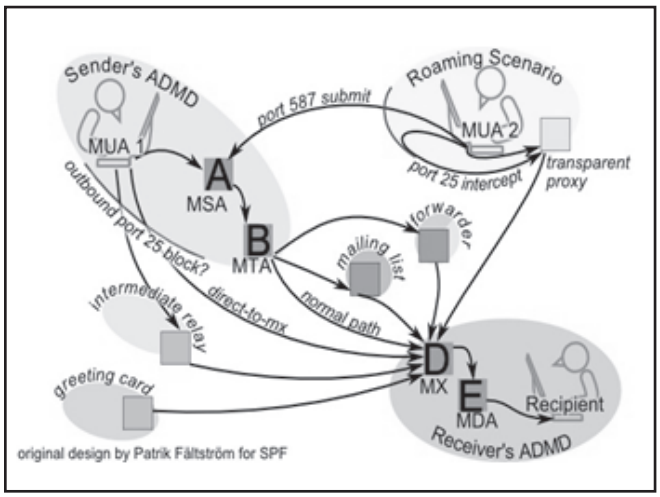

Fig. 2. A Schematic Representation of the Most Common Ways that an Email Message can get Transferred from its Author to its Recipient [12].

Fig. 2 shows that the most common way that an email message can get transferred from its author to its recipient. In this figure, a message submission agent (MSA) or mail submission agent is a computer program or software agent that receives electronic mail messages from a mail user agent (MUA) and cooperates with a mail transfer agent (MTA) for delivery of the mail. An MUA, mostly known as an email client, is a computer program in the category of groupware environments used to access and manage a user's email. A mail delivery agent or message delivery agent (MDA) is a computer software component that is responsible for the delivery of email messages to a local recipient's mailbox [12]. Our end-to-end solution is from MUA to MDA.

The process of sending an email is discussed next. An email header and body with signature components are demonstrated in Fig. 3:

1) User logs into an MUA, for example, from a webmail page or a mobile terminal application. A password *p* for sending email is required for this step. This password can be set same as the one user receives emails by default.

2) User inputs all header fields and body content of the email. The following steps are transparent to the user.

3) MUA application calculates hash value of the user's password. We can use the same mechanism used in Linux systems:

a) MUA chooses a salt value *s* that is a random data generated to combine with the original password, in order to increase the strength of the hash.

b) MUA chooses a hash algorithm *H* of index *i* from a pre-defined list, e.g., 1 for MD5, 2 for Blowfish, 3 for SHA-256, etc.

c) MUA calculates a hash value $h = H(p + s)$.

4) MSA application appends <*s1 = is$h*/> to the end of "data" component after message body, for example, <*s1 = 1Etg2ExUZ$Ds5e24NuQTP2tQ8vLn5Mw*/>means: using MD5 hash algorithm, *Etg2ExUZ* as salt, hash value of *H(p+s)* is *Ds5e24NuQTP2tQ8vLn5Mw* (*p*'s value is "emailpswd" in this example).

5) MSA uses *h* as key, and uses a predefined Message Authentication Code (MAC) algorithm, such as HMAC, to calculate a message tag *s2* of the message header *mh* and message body *mb* and the above *s1*: $s2 = HMAC_h(mh + mb + s1)$.

6) MSA appends *s2* to the end of "data" component, and sends it to MTA.
7) MTA first verifies authenticity of the claimed sender in email header:
a) MTA derives *i, s*, and *h* from *s1*, and a hash algorithm *H* of index *i* from the same pre-defined list embedded in the code.
b) MTA retrieves sending password *p* of the sender.
c) MTA calculates a hash value $h'=H(p+s)$, and compares it with the one *h* derived from *s1* in the message: $h'=?h$. If it is same, then continue; otherwise, reject the message.
8) If receiver is in the same domain as the sender, the message is forwarded as shown in the flow in Fig. 2.
9) If receiver is in different domain than the sender, MTA does the following:
(a) MTA calculates signature of *s2* by encrypting it with *SK_Sender*, the private key of sender's Administrative Management Domain (ADMD) email server: $s3 = E_{SK\_Sender}(s2)$.
(b) MTA appends *s3* to the end of message body.
(c) MTA appends *s4* — public key of sender's ADMD email server to the end of message body.
Correspondingly, the process of receiving an email is as follows:
1) MDA checks if the email is from the same domain. If it is, go to Step 4.
2) MDA reads signature *s4* at end of "data" component, gets sender's ADMD public key *PK_Sender*, and checks with well-known Certificate Authority (CA) to determine if it is authentic for sender's ADMD (more is discussed in Section VI). If not, reject the email and stop.
3) MDA reads signature *s2, s3* at end of "data" component, and verifies if *s3* is a valid signature of *s2* signed by sender's ADMD specified in *s4*, by decrypting *s2* with *PK_Sender*, and comparing the result with s3: $s3 =?D_{PK\_Sender}(s2)$. If not, reject the email and stop.
4) MDA reads the message header *mh*, message body *mb*, and signature *s1*.
5) MDA uses *h* in *s1* as key, and uses a predefined MAC algorithm, such as HMAC, to calculate a message tag *s2'* of the message header *mh* and message body *mb* and the above *s1*:
$s2' =HMAC_h(mh+mb+s1)$;
and compares it with s2 from the email: if $s2'=?s2$. If not equal, reject the email.

Notice that if the email is sent in the same domain, the MDA does not need to check authenticity of sender's ADMD server, but just checks if the sender uses correct username and password as claimed. Otherwise, MDA needs to verify if the email is really from the claimed domain. This is done by verifying the signature of the sender's ADMD server using its public key. The public key needs to be issued from a publicly accepted and trusted source, and can be verified from the source.

To summarize, the solution is an interlocked chain: it requires that the email sender provide correct username and password to generate *s1*, use *s1* as authenticator of the user who is sending the email, and generate *s2*; we use *s2* to protect integrity of the email and the authenticator, and generates *s3*; we use *s3* to protect integrity of *s2*; finally, we use *s4* to verify authenticity and integrity of

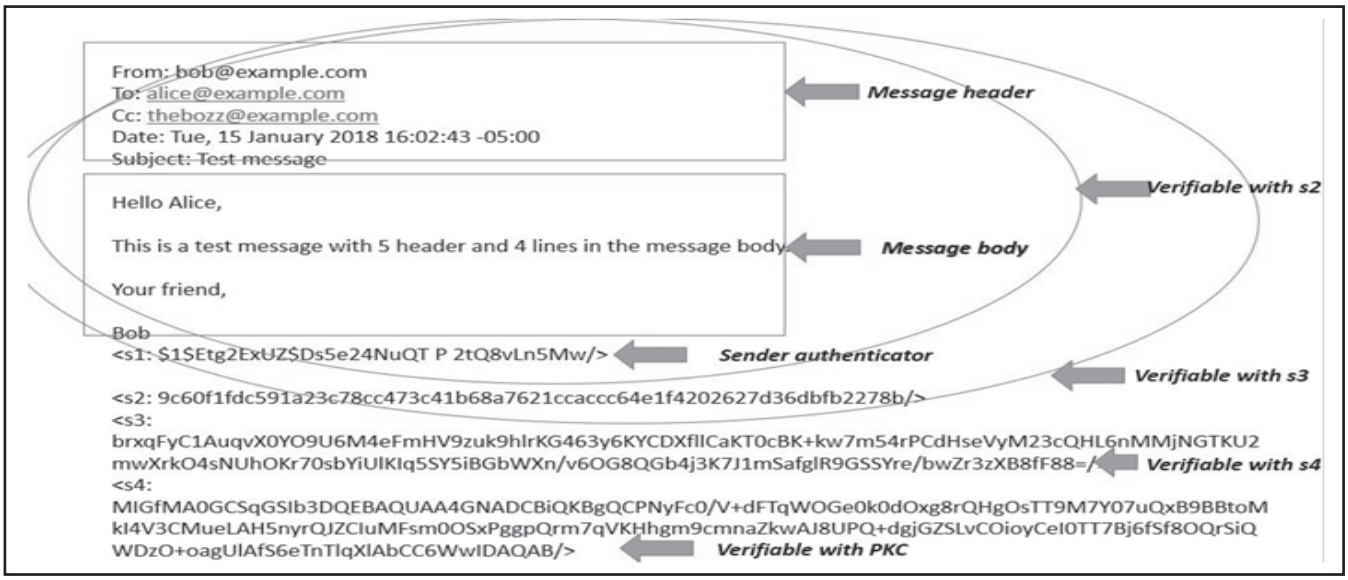

Fig. 3. An Email Header and Body With MAC tag and Signatures

s3, and *s4* itself can be verified with a public key certificate from a publicly accepted and trusted source.

V. SECURITY ANALYSIS AND PERFORMANCE EVALUATION

From the description of the solution, we can see that the security of the solution is based on "chain of trust" that is similar to the widely used "certificate chain", but the last segment employs existing technology and infrastructure. The verification of the sender is passed all the way down to the receiver. Since all the connections are interlocked with each other, the receiver and sender can establish end-to-end trust.

The security level of the solution depends highly on security of passwords and cryptographic algorithms chosen. For a secure hash algorithm or encryption/signature algorithm, the best attack to break is brute force. For this solution, if an attacker wants to forge an email with impersonated address, the effort in number of tries is as follows:

❖ *To generate a correct s1:* Assume the password uses letters in the ASCII table, excluding special characters, each letter in the password has 94 possibilities. If the password is 8-letter long, altogether the attacker needs 94^8, or approximately 2^{54} tries to generate the correct *s1*. If the password is 12-letter long, altogether the attacker needs 94^{12}, or approximately 2^{64}, tries to generate the correct *s1*.

❖ *To generate a correct s2: s2* is generated based on correct *s1*. The attacker needs to either break *s1*, or break the secure hash function used for *s2*. If SHA-256 is used in this step, the number of tries to break it is 2^{256}.

❖ *To generate a pair of correct s3 and s4:s3,* the signature on *s2* using private key of the sender's ADMD email server is used. The security of public/private key pair is determined by difficulty of integer factorization problem. The most efficient algorithm to solve it in some certain type of forms is *General Number Field Sieve (GNFS)* algorithm that takes $0(Exp^3 \sqrt{n})$ for *n*-bit number [6]. *n* is determined by the modulus *p* preset, such as 1024 or 2048.

The signature generation and verification will lower down performance of the sender's and receiver's servers. To evaluate the time consumption overhead brought by the signatures, I implemented the signature generation and verification algorithms with Java JDK 9.0 on Windows Server 2016 64-bit platform and Intel i7-6700 3.40 GHZ CPU/16G RAM desktop computer, and measured average run time for each of them. Core hash function used in the implementation is SHA-256, and public/private key algorithms used are RSA 1024-bit. The time spent on s1 generation or verification is approximately 50ms. The time spent on s2 generation or verification is approximately 60ms. The time spent on s3 generation or verification is approximately 50ms. The overall time overhead of signature generation and verification on both sender and receiver sides added on an email is less than 1s. Database access/retrieval time is determined case by case and is not included.

We see that the time overhead added to generate and verify the signatures is acceptable generally. On the other hand, we don't think the time overhead is a big concern, as email is not a time-sensitive communication measure, and a little percentage increase of process time does not matter much for normal users, but matters significantly for senders of batch spamming emails and phishing emails. So, this is a way to deter email attackers to some extent.

VI. DISCUSSION

I want to bring the following features and limitations of this solutions to the notice of readers:

A. End-to end Authentication and Integrity

The identity of the sender is verified and signed by sender's ADMD email server. The MAC tag and signature cannot be tampered or forged without detection. If User A is sending an email with name User B, MTA can detect the mismatch in sending password and rejects the email. If the sender's domain name is forged by the sender, MTA can also detect and reject it. If an email is intercepted and *s1* and *s2* are forged en route (for example, man-in-the-middle attack), it can be detected by receiver using sender's ADMD email server's signature, and public key certificate.

B. End-to-end Confidentiality

This solution is designed to ensure authenticity and integrity, by using MAC and signature. If end-to-end confidentiality is required, the sender needs to share a secret key with the receiver, or the sender needs to know the public key of the receiver, which needs to be distributed before creating the email and using it encrypt the email body. The email content is transparent to the solution. This means that an encrypted email body is treated the same way as a plaintext email body.

C. Security of Authenticator

If somebody gets your account password, s/he can send emails in your name — this is the normal case

understood by everybody. Password is not secure enough to ensure exclusive authentication. More reliable methods include hardware authentication token, biometric authentication such as fingerprint, iris, face recognition, voice recognition etc. that can prevent most of password-based impersonation. The proposed solution is compatible with these choices, as long as the sender's ADMD email server supports them.

D. Public Key Certificate Access

In the above description, we have the public key of the sender's ADMD email server attached in the email data. This is not efficient and adds traffic overhead, especially for frequent contacts. A better way is to adopt DKIM approach: have the certificates stored with DNS records of the sender's domain, and retrieve them when needed; a local certificate deposit is also suggested, as most web browsers do. This means *s4* is optional in the email.

E. s1 Through s4 Placement

s1 through *s4* can be added in extended headers of the email, but there is a concern that these extended headers might not be forwarded by the intermediate nodes between MTA and MDA. Technical details of this issue need to be considered and tested as future work.

VII. CONCLUSION

Spam emails and phishing emails are widely existing issues, and most of them are spoofed emails. We consider lack of end-to-end infrastructure as the main challenge when fighting against these issues. In the solution presented in this paper, we connect end users to existing security infrastructure, by having the sender's server verify and endorse its users, and passing this verification and endorsement to the receiver. The message tag and signature appended to the email provide authentication and integrity protection to the email. The specific contribution of this work is that it provides end-to-end security without end-user's intervention which is not seen in existing works. Since, there is no operation explicitly conducted by end users, it is much easier to deploy and use. The solution can be implemented on top of existing protocols as an optional component, without replacing email servers and routers in the internet infrastructure. All work is to be done on sender's and receiver's email client application and sever application.

I believe this is a novel, feasible, and promising solution in the current situation and in the near future.

REFERENCES

[1] S. Mundy, "Fraudsters' fingerprints on fake Samsung deal," *Financial Times*, October 11, 2013. [Online]. Available: http://www.ft.com/content/0b972892-3259-11e3-b3a7-00144feab7de

[2] J. B. Postel, "Simple Mail Transfer Protocol," August, 1982, IETF RFC 821.

[3] J. Klensin, "Simple Mail Transfer Protocol," October, 2008, IETF RFC 5321.

[4] T. Hansen, D. Crocker, and P. Hallam-Baker, "Domain Keys Identified Mail (DKIM) Service Overview," July, 2009, IETF RFC 5585.

[5] D. Crocker, "DKIM Frequently Asked Questions", Version: 16-Oct-2007 10:32. [Online]. Available: http://www.dkim.org/info/dkim-faq.html

[6] S. Kitterman, "Sender Policy Framework (SPF) for authorizing use of domains in email, Version 1," April 2014, IETF RFC 7208.

[7] M. Kucherawy and E. Zwicky, "Domain-based Message Authentication, Reporting, and Conformance (DMARC)," March 2015, IETF RFC7489.

[8] S. Ruoti, J. Andersen, D. Zappala, and K. Seamons, "Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client," *CoRR,* vol. *abs/1510.08555*, 2015. [Online]. Available: http://arxiv.org/abs/1510.08555

[9] D. Moolooand, and T. Fowdur, "An SSL-based client-oriented anti-spoofing email application," In *Africon Conf. Proc.*, September, 2013. doi: 10.1109/AFRCON.2013.6757757

[10] A. Zadgaonkar, V. C. Pandey, and P. S. Pradhan, "Authentication against e-mail address spoofing using application," *Int. J. of Sci. and Modern Eng.*, vol. *1*, no. *6*, pp. 13–17, May 2013.

[11] S. Zhao and S. Liu, "An add-on end-to-end secure email solution in mobile communications," in *Proc. of the 10th EAI Int. Conf. on Mobile Multimedia Commun., ser. MOBIMEDIA'17. ICST, Brussels, Belgium, Belgium: ICST (Inst. for Comput. Sciences, Social-Informatics and Telecommun. Eng.*, 2017, pp. 57–61. [Online]. Available: https://doi.org/10.4108/eai.13-7-2017.2270117

[12] P. Faltstrom, "Most common ways that an email message can get transferred from its author to its recipient," (n.d.). [Online]. Available: http://en.wikipedia.org/wiki/Email_authentication

[13] T. Kleinjung, "On polynomial selection for the general number field sieve," *Mathematics of Computation*, vol. *75*, no. *256*, pp. 2037–2047, 2006.

About the Author

Shushan Zhao received his B.Sc degree in Computer Science from Shandong University, China, and M.Sc degree in Telecommunication Software from Helsinki University of Technology, Finland, in 1992 and 2005 respectively. He completed his Ph.D. degree from the School of Computer Science at the University of Windsor, Canada in 2012. Before joining the University of Pittsburgh in 2016, Dr. Zhao worked as lecturer at Bishop's University and Vanier College in Canada. He also has rich experience in telecommunication and software industry, having worked as software developer at VMWare, Mitel, Ericsson, and Nuance, in Finland, and Canada.