MAILTRUST

ATTRIBUTE-BASED DYNAMIC ENCRYPTED EMAIL

by

MATTHEW CARLTON HUDNALL

SUSAN VRBSKY, COMMITTEE CHAIR
ALLEN PARRISH, CO-CHAIR
BRANDON DIXON
DAVID BROWN
TRAVIS ATKISON

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2017

ABSTRACT

Traditional email has significant security issues that have not been fully resolved by existing solutions. With regular email, content can be purposefully or inadvertently sent to users without adequate authority to view the content, and content can be forwarded on to other non-secure systems. While closed products and/or networks can address this problem, interoperability is severely lacking, as most solutions are not standards-based and result in isolated security silos. Without the ability to deploy a federated, heterogeneous architecture, users may simply fail to adopt a secure email solution – or be forced to inconveniently use a separate closed system for secure communications with legal requirements of high security (such as government classified communications).

Existing secure email solutions are severely lacking in their ability to meet the security needs of industry and government. Emails can be purposefully or inadvertently sent to users without adequate authority to view and content can be forwarded on to other non-secure systems. Encryption of content is troublesome and message revocation or altering is non-existent. The creation of large-scale secure email systems in their existing form has significant scalability concerns due to the implementation requirements of existing systems.

In this dissertation, we develop a secure email architecture that we call MailTrust. MailTrust provides a way to manage secure emails in an open network that guarantees that the recipient of a secure email has appropriate privileges to view the email, and that emails sent accidentally or intentionally to ineligible recipients are inaccessible by those recipients. MailTrust directly ad-

dresses the shortcomings of exiting solutions and provides a scalable architecture. We also examine the overhead and performance savings introduced by MailTrust and we document a prototype client/server platform that was created to test the feasibility of the architecture. In addition, the architecture could be easily extended to other domains. The MailTrust architecture provides a secure message exchange design that is applicable to many areas beyond just email. Future work could expand MailTrust to ecommerce, network access controls, remote code execution, and many other use cases where a secure dynamic trusted message exchange system is needed.

DEDICATION

This dissertation research is dedicated to my wife, family and God for whom I faithfully commit my life of service in the greater pursuit of their betterment. I am thankful to my friends, colleagues, and coworkers who all helped me throughout this process.

ACKNOWLEDGMENTS

I am grateful to my committee co-chairs, Dr. Allen Parrish and Dr. Susan Vrbsky, for all the extended time they spent helping guide me throughout this dissertation process. I would like to thank all of my committee members for their assistance as well. The UA Computer Science Department and its chair, Dr. David Cordes, have provided me with the knowledge necessary to navigate me through this research. I would also like to thank all of my colleagues at the Center for Advanced Public Safety (CAPS) who helped assist me in this work.

Above all, I would like to thanks my wife, children, friends, and God, without whom none of this work would have been possible.

LIST OF ABBREVIATIONS AND SYMBOLS

AD	Active Directory
AES	Advanced Encryption Standard
ABAC	Attribute Based Authentication
CA	Certificate Authority
CSR	Certificate Signing Request
DHS	Department of Homeland Security
DoD	Department of Defense
FBCA	Federal Bridge Certificate Authority
FCP	Federal Common Policy
FIdM	Federated Identity Management
FIPS	Federal Information Processing Standards Publication
GAL	Global Address List
GUID	Globally Unique Identifier
IA	Information Assurance
IDP	Identity Provider
IV	Initialization Vector
JSON	JavaScript Object Notation
JWT	JSON Web Token
MLS	Multilevel Security
NIST	National Institute of Standards and Technology

NSTIC	National Strategy for Trusted Identities in Cyberspace
PAP	Policy Administration Point
PBKDF2	Password-Based Key Derivation Function 2
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PGP	Pretty Good Privacy
PIP	Policy Information Point
PKI	Public Key Infrastructure
RBAC	Role-based Access Control
SAML	Security Assertion Markup Language
SC	Security Category
SEM	Secure Email Client
SMTP	Simple Mail Transfer Protocol
SP	Service Provider
SSL	Secure Sockets Layer
TD	Trustmark Definition
TD Spec	Trustmark Definition Specification
TIP	Trust Interoperability Profile
TM	Trust Management
XACML	XML Access Control Markup Language

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

Email is generally regarded as an insecure method of electronic communication for numerous reasons. Most notably, the default does not guarantee the authentic identity of either the intended sender or receiver of a message, nor does it guarantee the confidentiality and integrity of the message. While these problems can partially be addressed with commonly utilized technologies involving certificates and email client plug-ins, current practice is insufficient for high-security applications, such as classified communications among clients of different email systems. The research presented in this dissertation leverages "Trustmarks" that were developed primarily to support efficient single sign on in a federated environment. The goal of Trustmarks is to support secure emails between multiple systems where there are particularly stringent confidentiality and integrity requirements. Such a system could increase the ability of users at disparate organizations to communicate without fear that sensitive information might intentionally or accidentally be disclosed. Although there are many barriers to adoption, such a system might also eventually reduce the reliance on separate communication networks and systems for classified communications.

Secure email has existed in many forms since the X.509 based Public Key Infrastructure (PKI) was introduced in 1988 and Pretty Good Privacy (PGP) was released in 1991 [1], [2]. Standard email uses a store-and-forward system from sender to recipient that has traceability issues, and centralized management of the ecosystem is impossible due to the fragmented nature of

the systems [3]. Secure Sockets Layer (SSL), which is commonly used to protect online purchases and website communication, can only be applied to the email transmission layer, and thus, it cannot protect emails at rest [4].

A number of high-profile incidents surrounding secure email have prompted new research into securing this long-standing communication medium. There have been multiple cases where US Secretary of States have run their own personal email servers that were operated outside of federal control [5]. Since their servers were not part of the federal PKI infrastructure, they had no method to verify the authenticity of emails that were received.

There are multiple benefits to running a separate email system including: (1) convenience of being able to choose a solution that supports more devices, and (2) maintaining control of the flow of information. Government-run email systems are subject to Freedom of Information Act (FOIA) requests including inquiries by oversight/adversaries/journalists/investigators, but personal email systems are not. By having one email system that is under an individual's control, it can be used for personal communication without fear that future FOIA requests might intrude on their personal life. The problems arise, however, because those personal email systems are not up to government standards, and they cannot provide the appropriate safeguards for sensitive information.

The pervasiveness of email as a communication medium in our society presents an easily targeted platform. For almost a year between summer of 2015 and July of 2016, hackers had access to private Democratic National Convention (DNC) systems [6]. Emails containing highly protected DNC secrets were released by the hackers through wikileaks.com. The fallout from the release of the email content resulted in the resignation of the DNC chairperson and caused considerable embarrassment across the breadth of that political party.

Cyber warfare is becoming a common battlefront, and email systems represent a soft target with a wealth of information. Without further protection and modernization of the security surrounding our email systems, it is only a matter of time before additional attacks on email systems result in even greater damage. There have been many attempts in the past to propose secure email solutions, but each has fallen short due to either the complexity of the implementation or lack of a complete solution set for the scope of the problem.

This dissertation will outline a new solution called MailTrust that builds upon solid concepts from past research. It takes security in email to a much higher level by incorporating new technologies and methods that enable MailTrust to succeed where other systems have fallen short. MailTrust defines a new email security and encryption process based on AES encryption, and it defines a new use model that extends Trustmarks to the domain of email. The use of Trustmarks allows email non-repudiation controls, dynamic entity management, a secure email content information process, and identity coordination with external organizations. MailTrust provides message revocation through single-source secure message repositories, and a XACML-based email document security model for content control. In Chapter 2, a comprehensive email overview is provided to document the advantages and deficiencies across existing solutions. We examine the existing landscape of secure email and compare MailTrust to those production systems as well as systems that have been proposed in literature. In Chapter 3 we discuss the trusted identity concepts that govern MailTrust and provide detailed background on the Trustmarks foundation on which MailTrust is built. Chapter 4 presents the security and control measures that are available in MailTrust including message revocation, message tracking, and a hierarchical document security model. Chapter 5 documents in detail the prototype MailTrust system that was created to test the architecture. Chapter 6 examines the performance simulations

that were run against the prototype system to determine the benefits and overhead associated with the design. Finally, Chapter 7 summarizes the MailTrust work and presents future areas of research that could be explored to increate the adoption of a MailTrust system and to expand MailTrust to other domains.

CHAPTER 2 - LITERATURE REVIEW

The large-scale adoption of secure email has been hindered by a lack of integration in the sending of SMTP protocols, and the widespread client retrieval protocols such as POP3, IMAP, Exchange, etc. Most implementations of secure email systems involve the encryption of a message and the cryptographic signing of the encrypted message; but, some closed-loop systems operate on a "depot" model whereby the message content resides centrally, and notifications of new messages are propagated to mail clients.

In the following sections, we document the current email security practices adhered to by the Department of Defense (DoD), including details of the affiliated control measures. We then cover a number of secure email systems that have been either prototyped by researchers or implemented in industry-provided solutions. An overview of Public Key Infrastructure (PKI) is also included, since PKI provides the most widely-used implementation of email signing and encryption. Issues across all of the aforementioned systems and domains are discussed. Finally, we introduce MailTrust, a secure email architecture designed to fill the large gaps that exist with today's most secure email systems. We note that while this research primarily targets the use of MailTrust in the context of government/DoD systems, the overall architecture is beneficial to industry and individuals as well who wish to utilize the benefits of secured messaging.

The United Stated Intelligence Community (IC) operates under the Office of National Intelligence that is headed by the Director of National Intelligence (DNI), who reports to the President of the United States [7]. Within the IC are 16 member agencies that include the

FBI, CIA, NSA and other key intelligence partners. The Department of Defense (DoD) controls six of those member agencies including the Marine Corps Intelligence, National Geospatial-Intelligence Agency, National Reconnaissance Office, Office of Naval Intelligence, Defense Intelligence Agency, Twenty-Fifth Air Force, Intelligence and Security Command, and the largest US security agency: the National Security Agency [8].

Sharing information in a timely but secure manner has been a critical need within the intelligence community, but is also one that has substantial room for improvement [9]–[11]. The Federated access and control of information within the Intelligence Community has brought forth a number of standards [12], [13], but those standards do not address the legacy IT "backdoor" that email has created within these systems. The large-scale adoption of secure email has been hindered by a lack of integration in the sending SMTP protocols and widespread client retrieval protocols such as POP3, IMAP, Exchange, etc. Most implementations of secure email systems involve the encryption of a message and the cryptographic signing of the encrypted message, but some closed-loop systems operate on a "depot" model whereby the message content resides centrally and notifications of new messages are propagated to mail clients.

Data breaches from the inappropriate dissemination of confidential/secret/top secret information has been at the forefront of many recent highly public security incidents [5]. The ease of which secured content can be purposefully or inadvertently sent to persons without clearance through email highlights the need for increased security measures [14]. Additionally, security around email accounts themselves as storage repositories of potentially sensitive information has been brought to light lately as well. Billions of email accounts have recently

been compromised by hackers [15] and the content of those accounts could have sensitive data that may have been sent to those non-secure accounts.

In 2015 Ruoti et al [16], a basic feature set of secure email was defined as follows:

- Message confidentiality: only the dialogue parties are privy to the message
- Message integrity: the message was not tampered with
- End-point authentication: the dialogue parties have verified identities and/or credentials.

These properties (confidentiality, integrity and end-point authentication) are critical to the success of any secure email system. Current secure email systems implement these properties to a limited extent and existing research addresses them to varying degrees. In the context of these properties, we are proposing the following research questions to guide the design of our MailTrust architecture:

- What are the current secure email technologies that exist?
- To what extent are these technologies sufficient to meet the properties of confidentiality, integrity and end-point authentication?

Intelligence information is typically "classified" according to its sensitivity and the degree to which it can be shared. Since the DoD is responsible for the largest security agency as well as the largest number of agencies, the DoD's security policies for annotating sensitive information in email is the first step to understanding the semantics of messages that must be sent through a secure email messaging system.

The next section of this chapter describes the current practice with regard to DoD message annotation. This is followed by a section that discusses the ubiquitous use of PKI to

implement secure email. While this is an obvious first step, PKI addresses confidentiality and integrity, but does not address the end-point authentication that is critical for sharing classified intelligence information. A section is then dedicated to presenting several systems that have generally been introduced to address the end-point authentication side of the equation. The final section of this chapter contains a discussion of the limitations of these systems and a proposal for future work.

Intelligence Information Message Annotation

The DoD manual number 5200.01 Volume 2 [17] specifies the policies and procedures for marking of classified information. The control procedures specified within denote the document marking techniques that all branches of DoD are to use to appropriately tag information within a document. As required by the DoD directives referenced in the manual, all classified information shall be identified clearly by marking, designation or electronic labeling in accordance with the parameters specified in the manual. These markings are meant to serve the following functions:

- Alert holders to the presence of classified information.
- Identify, as specifically as possible, the exact information needing protection and the level of protection required.
- Give information on the source(s) of and reasons for classification of the information.
- Identify the office of origin and the document originator who applied the classification markings.
- Provide guidance on information sharing, and warn holders of special access, dissemination control, or safeguarding requirements.
- Provide guidance on downgrading and declassification for classified information.

According to the manual, the security levels of the content are based on the following levels:

- SF 710 – Unclassified
- SF 708 – Confidential
- SF 707 – Secret
- SF 706 – Top Secret.

A "banner line" at the top of the document denotes the overall classification of the document including the most restrictive control markings applicable to the overall document. For example, if any portion of the document contains "Secret" level information and there is no "Top Secret" information within a document, then the banner line would represent "Secret" as the overall document classification level.

These security levels are meant to be applied to all documents relating to foreign or domestic national security, and they are further extended by dissemination control parameters specified within the document. Dissemination control is based upon "REL" or "REL TO" markings that denote the parties to which the document has been authorized for release. A document marked "REL TO USA, GBR" would be authorized for release to domestic US-based agencies and to agencies in Great Britain.

The banner line for dissemination control also represents an aggregate for the overall discrimination control parameters for that given document. Suppose subsections contained within a document restricted certain content to REL TO USA, GBR, CAN (for release to USA, Great Britain, and Canada, respectively), but another section within the document contained discrimination control tags that restrict that area to REL TO USA, GBR. The overall document control restrictions would be REL TO USA, GBR since that represents the

least privileged control group that is allowed to see all of the information within the document. As seen in *Figure 2.1*, these document tagging techniques can be merged together to provide finite level document control within a given work product.

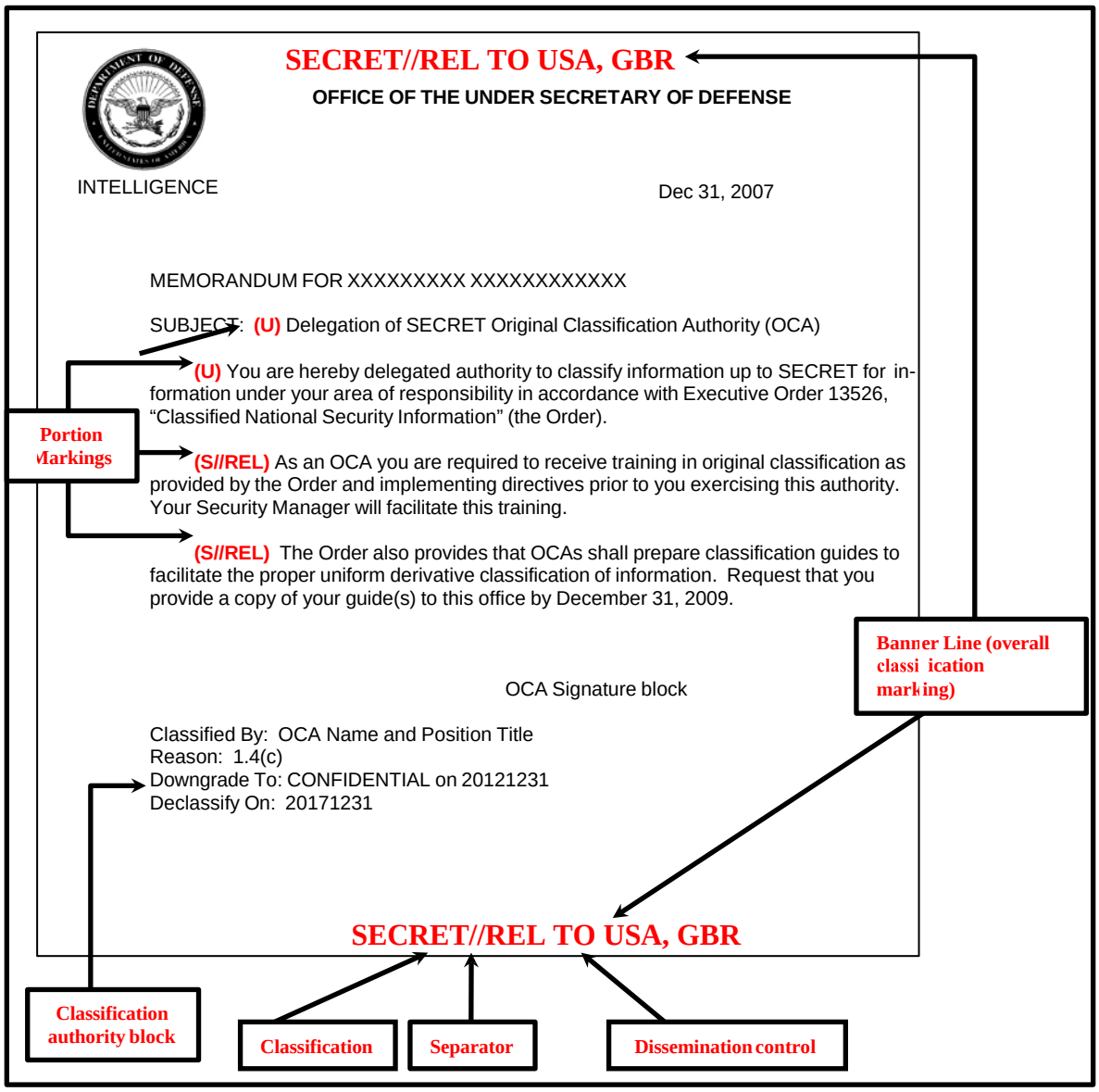


Figure 2.1 - Example of DoD Document Classification Markings [18]

UNCLASSIFIED – CLASSIFICATION MARKINGS FOR ILLUSTRATION PURPOSES ONLY

DoD 5200.01 Volume 2 applies not only to physical documents but also applies to email messages. Section 17(b) of Enclosure 3 within the manual calls out the procedures

required for document tagging of email messages. The section notes that email transmitted on classified systems or networks "shall display the banner line at the top and the bottom of the body of each message as a single linear text string showing the overall classification, including dissemination and control markings." The subject line, as well as attachments to the email message, are required to be marked in accordance with the appropriate security level of the content. As seen in *Figure 2.2*, an example email message contains the same markings as a regular DoD document.

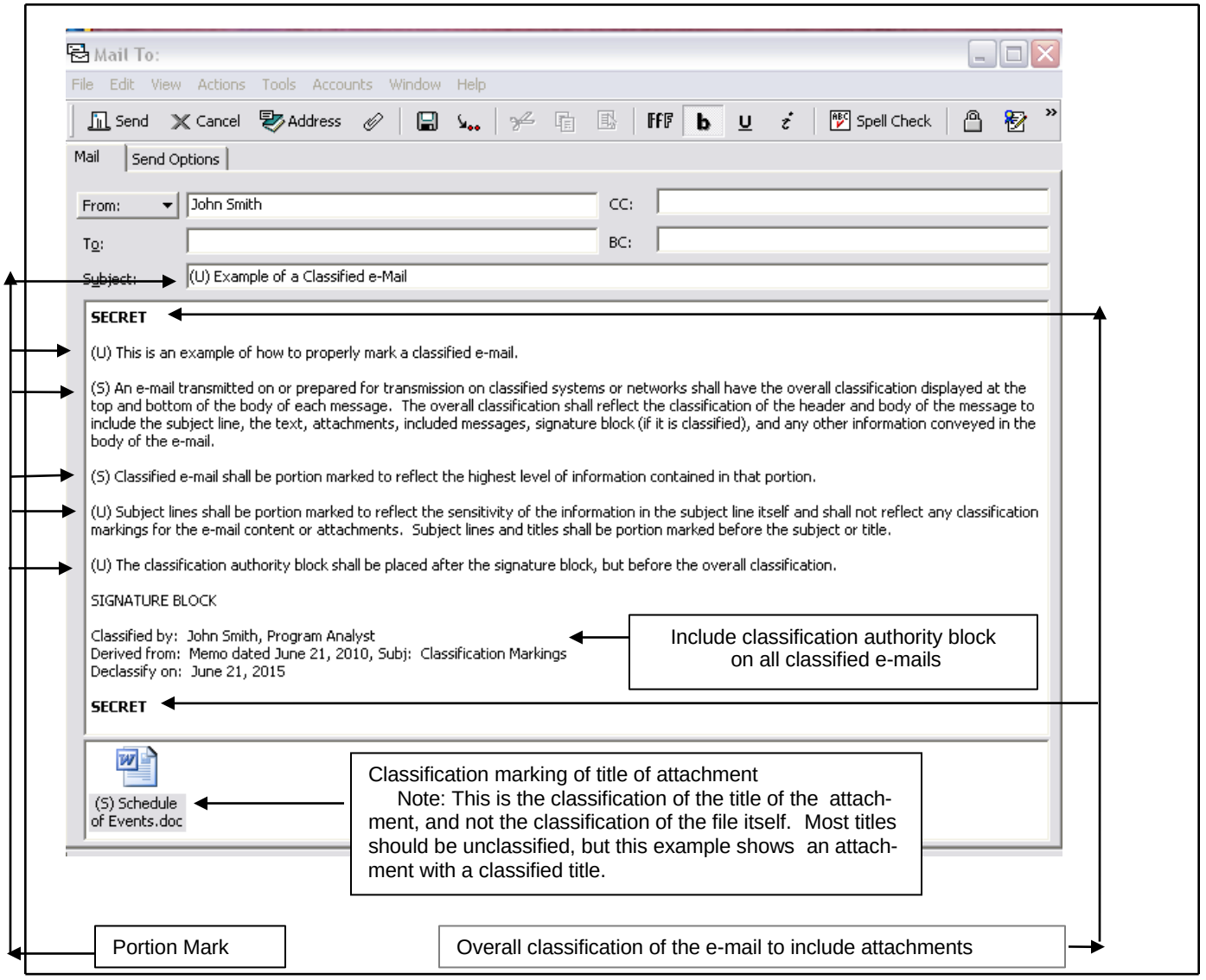


Figure 2.2 – Example of DoD Email Classification Markings [18]

UNCLASSIFIED – CLASSIFICATION MARKINGS FOR ILLUSTRATION PURPOSES ONLY

The manual notes that while some organizations may use automated tools to mark electronic messages (emails, texts, instant messages, etc.), it remains the individual's responsibility to properly mark classified messages (including banner marking, portion markings, and classification authority block) when an automated tool is used.

Public Key Infrastructure (PKI)

PKI can be defined in multiple forms but in the context of email security, PKI is based on the hierarchical Certificate Authority (CA) model. At the base of the hierarchy is the root certificate authority that acts as the centralized source of all subsequently discriminated authority. Digital certificates, which are commonly in the X.509 certificate format, are issued by CAs within a PKI ecosystem. A certificate contains a number of items that are key to its functionality and those fields are outlined in Table 2.1. Digital certificates, usually in the X.509 format [19], [20], are issued by CAs and are have a tree-based hierarchy whereby a root certificate authority, root-CA, creates an initial certificate and then digitally signs every certificate that it issues. Any child-CAs after the root-CA that issue certificates do so in a tree model with the parent CA's signature at each level. This inheritance model allows for centralized revocability as well as simplified trust, since policies can be set, such as "trust everything issued by root-CA X."

Table 2.1 - Components of an X.509 PKI Certificate

Field	Purpose
Serial Number	Used to uniquely identify the certificate
Subject	The person or entity identified
Signature Algorithm	The algorithm used to create the signature
Signature	The actual signature to verify that it came from the issuer
Issuer	The entity that verified the information and issued the certificate
Valid-From	The date the certificate is first valid from
Valid-To	The expiration date
Key-Usage	Purpose of the public key (signature, certificate signing, etc.)

Public Key	The public key
Thumbprint Algorithm	The algorithm used to hash the public key certificate
Thumbprint	The hash itself; abbreviated form of the public key certificate

In issuing an X.509 certificate in a PKI model, an asymmetric key pair is generated that serves as both a signing and verification mechanism for subsequent transactions. The private key is used to sign the original Certificate Signing Request (CSR) issued by the CA to the requesting party, and it is then handed off to the issuer. It is not stored on the CA. The public key is part of the certificate itself and is stored in the thumbprint section of the certificate.

Let us suppose that Bob wants to email Alice using PKI, and Bob chooses to digitally sign the email message to ensure the authenticity of the message. By choosing this option in his email client, the email client extracts out Bob's private key from Bob's certificate store on his machine and computes a hash of the email using the hash algorithm defined in Bob's certificate. The hash of the message is then appended onto the email message in a header section that is separate from the email body. When Bob emails the message with the signed content, a copy of Bob's public certificate is also included with the email as shown in *Figure 2.3*. When Alice receives Bob's email, Alice's email client first verifies the authenticity of the certificate attached to the email by a certificate trust chain from a common root certificate authority. If both Bob and Alice have a common root certificate authority in their machines' certificate trust stores, then certificates issued by that CA (or CAs) under that root are trusted cryptographically by both Bob and Alice.

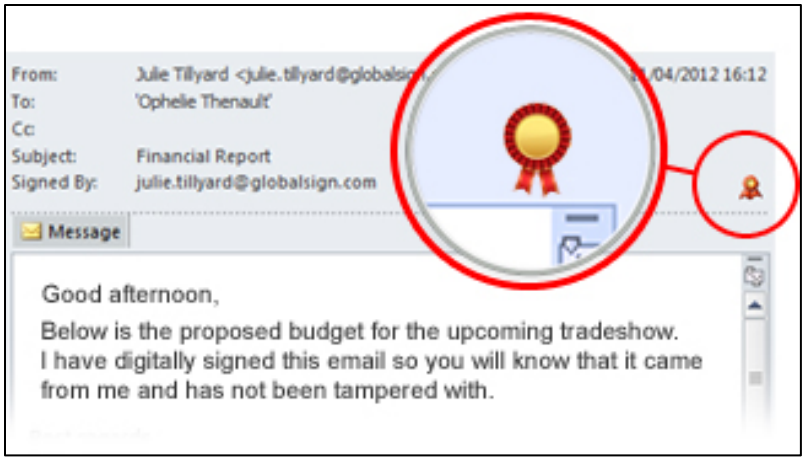


Figure 2.3 - PKI Certificate Signed Email [21]

After the certificate is verified by the client, the email client uses the public key in the certificate along with the thumbprint algorithm specified in the certificate to re-compute the signing hash for the email. If the hash matches the hash specified in the signing header of the email, then the content of the email is deemed authentic by the email client, and the email client shows a certificate icon indicating that the authenticity of the email content has been cryptographically verified through a PKI validation model. As seen in *Figure 2.4*, the verification of a PKI email is practically transparent to the user if the signing certificate used by the sender has a corresponding root certificate in the receiving user's certificate store.

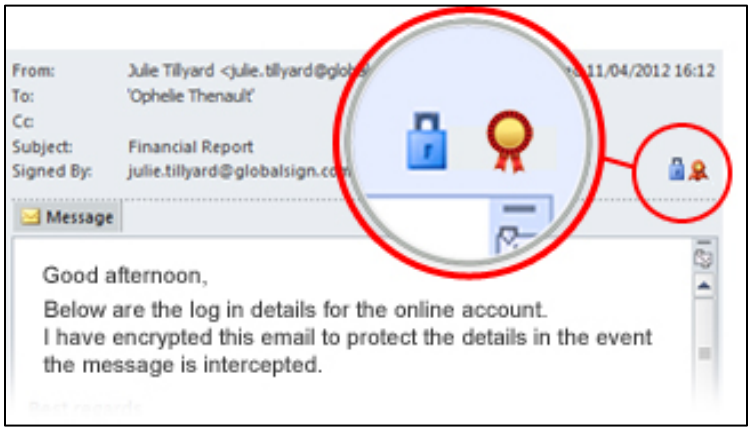


Figure 2.4 - PKI Certificate Signed & Encrypted Email [21]

Email encryption via PKI is a slightly more complex process, and this has reduced its adoption rate accordingly. In order for users to send encrypted emails to other users, they must first have the public key of the intended recipient. In single organizations, this is commonly handled through the use of a Global Address List (GAL). Public keys for users within an organization are published to a centralized GAL and that public key is then picked up by an email client that is trusted by the GAL. The common pieces involved in this are Microsoft Exchange email server coupled to a Microsoft Outlook email client that uses a connection to the GAL that is stored in the organization's Microsoft Active Directory (AD). An obvious disadvantage of this type of system is the reliance upon a single vendor's technology stack to facilitate the encryption of email messages in an organization.

Once the sender's email client has access to the intended recipient's public key, either through the GAL or from a direct exchange a priori with the user, then the sender's client uses that certificate to encrypt the email message using the public key and the algorithm in the certificate. Since the PKI key pair originally generated by the CA is asymmetric, only the recipient who has the corresponding private key will be able to decrypt the message. The message is encrypted by the recipient's public key and then signed by the sender's private key to ensure the authenticity of the author, and then the encrypted message with the signed body and sender's certificate are sent to the recipient. Once received, the recipient's client verifies the sender in the same manner previously described, and then the recipient's email client extracts the recipient's private key from the user's certificate store and uses that key to decrypt the content of the message.

X.509 certificates used in secure email are based on an asymmetric key pair that is generated at the time of the certificate issuance. A public/private key pair allows for the distribution of

the public key while keeping the private key secure. Transactions of all types can be cryptographically signed by the private key and verified by the public key and data can be encrypted by the public key and decrypted only by the private key. The private key is used to sign the original Certificate Signing Request (CSR) that is issued by the CA to the requesting party and then the certificate and keys are passed on to the issuer. The private key is not centrally stored on the CA, but since the certificate is issued and signed by the CA, it can be centrally revoked/cancelled. The public key travels with the certificate in the "thumbprint" area of the cert so that users can utilize it in transactions to verify authenticity of the party with which they are communicating.

S/MIME (Secure/Multipurpose Internet Mail Extension) utilizes the features of PKI to provide email non-repudiation and encryption. Emails are signed with a private key and sent to a recipient. The recipient is then able to verify authenticity of the sender using the sender's public key. Email encryption is also allowed in S/MIME, but it requires a priori key exchange of the recipient's public key. If a sender has a recipient's public key, then they encrypt that message with the recipient's public key; and then only the recipient is capable of viewing the message since only the recipient has their private key.

PKI email signing and encryption is a great step forward beyond insecure and unverifiable traditional email, but it fails to address a number of concerns, and it is complex in nature. The requirement for a sender to have the recipient's public key beforehand has vastly limited the use of email content encryption. This is because the silos of users cannot easily share public keys since there is limited connectivity between GALs; and the requirement of a single vendor technology stack also limits cross-platform usage.

Another fundamental limitations of email PKI is the lack of content dissemination control measures. Email PKI allows for the verification of a sender, or in certain circumstances the verification of the recipient and encryption of the message; but, it has no control measures in place to address the control of authority to view the information that is sent. There are no controls in place to verify the security of a recipient's email server, the security of the identity management system of the recipient, or, more broadly, the authority of the recipients to view the content that is sent to them. Inadvertent information leakage, whereby an incorrect person was emailed, still exists in PKI email, and there are no controls in place to evaluate the status of a user. It is assumed in the PKI email model that if recipients have access to their emails, then they have the current authority to view the content. PKI has no verifications controls in place to address: (1) if a recipient's security clearance has expired, (2) if they have been terminated, or (3) if they never had the appropriate clearance to view the information sent.

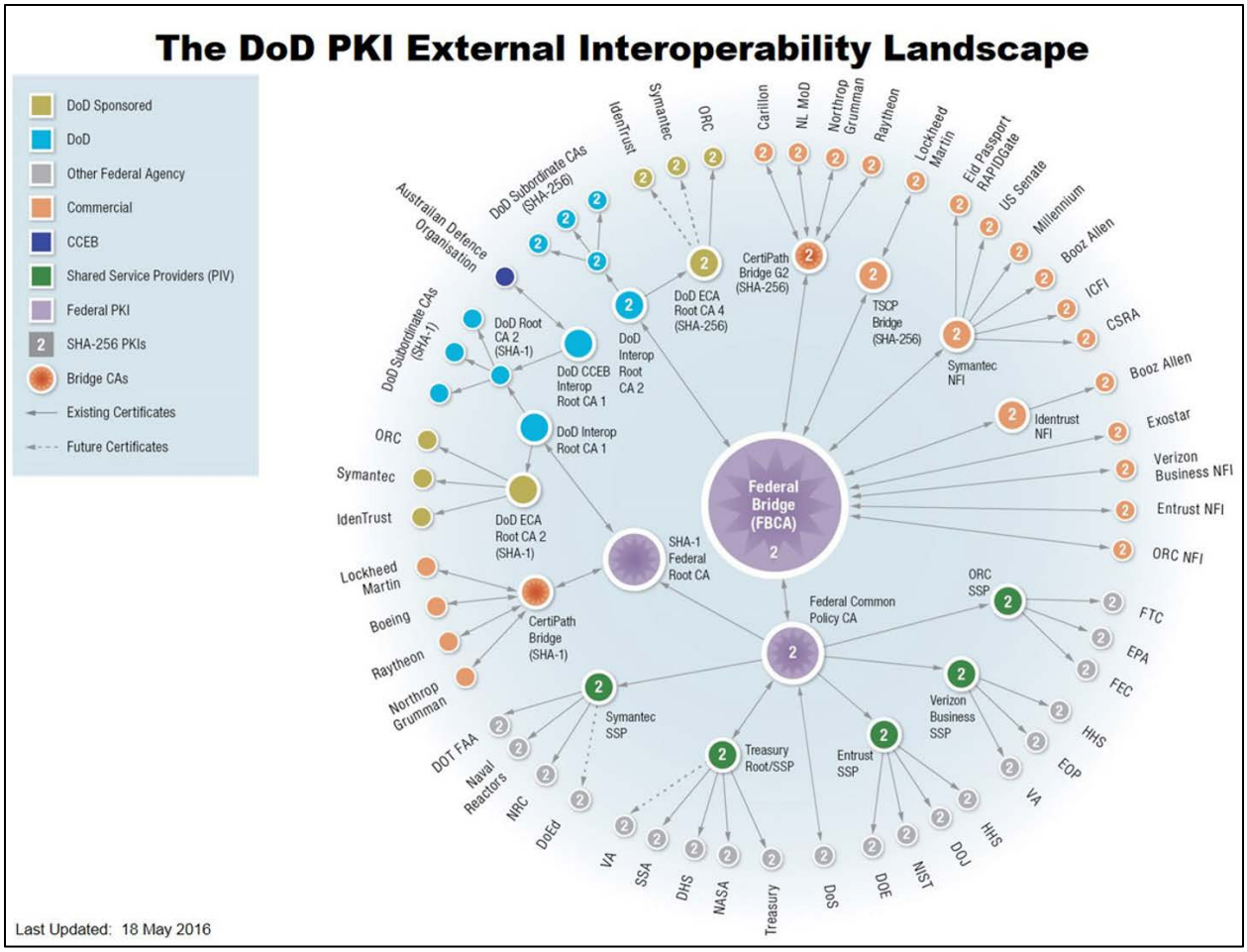


Figure 2.5 - DoD PKI Landscape [22]

DoD currently relies upon a PKI infrastructure that is built upon a Federal Bridge Certificate Authority (FBCA) to connect together DoD branches, partner agencies, contractors and other entities with which it wishes to leverage interoperability between their associated email ecosystems. As seen in *Figure 2.5*, this PKI infrastructure is a hub-spoke environment with multiple federal bridge nodes at the center. This connectivity allows, for example, Department of Homeland Security (DHS) personnel to author an email message and send the email to persons at affiliated contractors like Lockheed Martin. The recipient at Lockheed can verify the authenticity of the message since that user has certificate authority traceability back to the root Federal Common

Policy (FCP) certificate authority. Since the sender's certificate was issued by a certificate authority that is chained back to the FCP (and the recipient trust certificates issued by CAs that chain to the FCP), then a trusted pathway is established.

There are multiple problems, though, with the current PKI system under which DoD and other federal agencies operate. First, the authenticity of an email is guaranteed only within the context of the verification of the validity of the individual, and this does not provide meaningful background as to what authority the sender possesses. An example of this would be if two people at DHS have similar names. If one DHS user has an email address of john.m.smith@dhs.gov and another DHS user has an email of john.n.smith@dhs.gov then these are both valid accounts; but in this example understanding the authenticity of the sender only extends to the personal knowledge a recipient possesses as to the middle name of the sender. Both parties will appear through the federal PKI system as being valid. Given that there are approximately 2.9 million federal employees, minimal name deviations can play havoc in ensuring the identity of individuals.

In addition to name similarities, the existing federal PKI infrastructure does not allow for dissemination control parameters beyond what the users are able to manually enforce. Since the DoD email document control scheme is based purely on text-based document tagging, it is at the sole discretion of the user as to whom they copy emails and/or to whom they forward content that they have received. We interviewed multiple DoD personnel during the course of this research to determined if real-world practice was in conformance with the DoD specified security requirements. While all interviewed said that they were trained to tag message content sections individually, in practice, the consensus was that due to time constraints, documents are summarily tagged with the security level of the most restrictive content contained within the document. Additionally, the forwarding of emails and the sending of email content outside of the federal PKI infrastructure

loses any traceability and control of the sensitive information, since it has left the federal ecosystem.

Existing secure email systems

In 2015 Ruoti et al evaluated the usability of three predominant secure email systems, Pwm, Tutanota, and Virtru, and they determined that participants overwhelmingly chose the secure email system that was integrated within their existing email systems over those that require a secondary account (email depot model). The study concluded that, with appropriate training, the users had a high system usage/adoption rate. The researchers also investigated systems based on PGP, but they concluded that PGP implementations had low usability, and thus, they did not include them in their study. A basic feature set of secure email has been defined as follows:

- Message confidentiality: only the dialogue parties are privy to the message,
- Message integrity: the message was not tampered with, and
- End-point authentication: the dialogue parties have verified identities and/or credentials.

These properties (confidentiality, integrity and end-point authentication) are critical to the success of any secure email system. Intelligence information is typically "classified" according to its sensitivity and the degree to which it can be shared. Since the DoD is responsible for the largest security agency as well as the largest number of agencies, the DoD's security policies for annotating sensitive information in email is a critical step towards understanding the semantics of messages that must be sent through a secure email messaging system. (The next section in this chapter describes the current practice with regard to DoD message annotation.) While PKI addresses confidentiality and integrity, it does not address the end-point authentication that is critical for sharing classified intelligence information.

The Simple Mail Transfer Protocol (SMTP) does not contain a method by which a sender's identity can be verified. This has created an environment that is inundated with false email phishing scams and also creates a legitimate security concern for corporate and government entities. The current standard for secure email centers around the Secure/Multipurpose Internet Mail Extensions S/MIME protocol. S/MIME relies on a PKI infrastructure through which two message-exchanging email clients have a common trust pathway back to a Certificate Authority (CA). Non-repudiation in S/MIME is maintained as long as the private message signing key is not compromised [23], [24].

As a result of a case study, Kapadia in [25] notes that "more sophisticated protocols for non-repudiation are needed" and that S/MIME does not work in practice for casual users. Violating the integrity of email messages in today's email ecosystem is extremely simple. The CA trust chain is commonly degraded by pre-installed third-party CA certificates that have not undergone thorough auditing procedures [26]. The lack of vetting produces a weakest-link scenario that attackers can exploit by obtaining a certificate at the CA that has the lowest barrier of entry. Since the average user does not understand the trust framework that makes up a PKI system, they are often left with a false sense of security when implementing PKI through S/MIME.

The authors, Chen and Ma, noted in [27] that PKI-based security frameworks such as S/MIME are not efficient due to the high degree of management associated with certificates. PGP was examined as well, but it was deemed only suitable for small-scale groups. An identity-based system was proposed by Chen and Ma that relies on a centralized server for encrypting email content. The enforcement of security paradigms in computer systems based off of identity attributes dates back to Saltzer & Schroeder's foundational work in 1975 [28]. Adi Shamir in 1984 first

proposed an identity-based public key architecture [29], and Boneh & Franklin produced a practical IBE solution in 2001 based on Shamir's work [30]. Baldwin expanded the work further in 2002 [31]. In the Chen and Ma model, a centralized server facilitated the email encryption, and the encrypted content was transferred over the network. The authors noted that Role-Based Access Control (RBAC) is ultimately needed, but that was not implemented currently in their model.

In 2006, Masone proposed the Attribute Based Usefully Secure Email (ABUSE) system [32]. It is a decentralized nonhierarchical PKI system that uses PKI to control trust relationships between individuals. The authors noted that S/MIME is based on X.509 Identity Certificates that are issued by centralized Certificate Authorities (CA). One key issue with existing S/MIME implementations is that root CAs are all treated as equal. So if the vetting procedures are lax at any of the companies that issue the certificates, then a user can exploit that weakest link to gain an identity that would be as equally trusted as those provided by more thoroughly vetted CAs.

An analysis of "unfamiliar correspondents" was examined in [32] to classify types of emails where the recipient might not know the identity of the sender. The authors point out that in large organizations, it is increasingly likely that a recipient does not know the identity of the sender. Cases of same names, similar names, and spoofing of identities were given as examples of issues surrounding the identity of senders. In most cases, the authors note, users revert to manual verification by phone, searching of websites/directories, or just assuming that the sender is legitimate. This results in attack vectors that are regularly exploited by hackers.

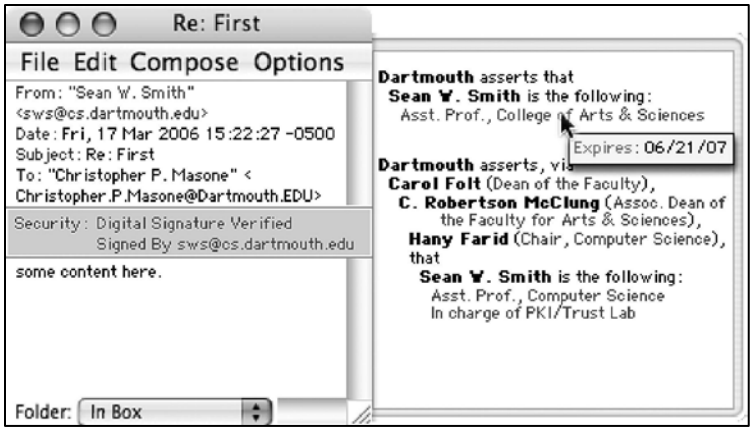


Figure 2.6 - Example ABUSE System Trust Management Insight Tool [33]

To solve these issues, Masone and Smith discuss implementing a Trust Management (TM) system on top of email. This access control system is implemented by choosing a policy language, attaching credentials to signed messages, and creating a policy enforcement engine for onboard users to accurately capture their trust behaviors while viewing emails. They note that TM is too complicated for recipients in its standard form so they proposed the ABUSE system. The ABUSE system replaces the TM framework components with attribute-based bindings. The ABUSE system presents human-readable identity assertions via a modified email client to the recipient as shown in *Figure 2.6*. As Paci et al. stated in [34], the information disclosed during a transaction should be limited to the scope of the transaction at hand. Finding the appropriate balance between information disclosure to prove identity, as suggested in the ABUSE system, and the minimum required to convey authenticity of identity is an optimization problem that needs to be further researched.

The authors noted that their system could likely be implemented using SAML, but since their custom application implemented the required logic, that option was not pursued. To implement ABUSE across multiple organizations, the authors sighted the use of Bridge Certificate Au-

thorities as a means of joining together organizations. While this bridge method is what the Department of Defense relies upon with the "Federal Bridge," it creates a web that can be challenging to manage, and it has multiple core vulnerability points.

As an alternative to traditional PKI, Blaze et al. propose a policy-based architecture called PolicyMaker [19]. PolicyMaker wraps standard PKI signing into a rule-based infrastructure. An email system based on PolicyMaker was proposed whereby recipients would have rule-based email evaluators to distinguish junk mail (or fake mail) from authentic emails. Policies, such as:

> policy ASSERTS pgp:"Oxf0012203a4b51677d8090aabb3cdd9e2f" WHERE PREDICATE=regexp:"(From: Alice) && (Organization: Bob Labs)" ;

would indicate that the email client wanted to see and trust emails from Alice who works at Bob Labs and had a corresponding PGP signature. This model works as a baseline trust mechanism, but it does not address the problems of connectivity and scalability between organizations. Further, there is no vetting mechanism in place to verify the assertions contained within the statements.

The CryptoNET system [35] is a hierarchical PKI-based system for secure email transmission that relies on intermediary Secure Email Client (SEM) servers for content encryption and decryption. As shown in Fig. 7, the CryptoNET proxy-based architecture extends the traditional email security model by adding Policy Decision Points (PDP) and Policy Enforcement Points (PEP) at both ends of email transmissions that enforce authorization policies upon content attached to and within the emails. CryptoNET implements a Policy Administration Point (PAP) to digitally sign email headers to ensure the authenticity of the documents and to prevent email spoofing/forgery.

The CryptoNET model relies on a distributed key architecture since the receiving servers have to be able to decrypt the incoming message content. This creates a system-wide vulnerability in that if any node of the ecosystem is compromised, then the decryption keys for all content would be compromised. Additionally, the CryptoNET system does not include measures for addressing policy enforcement by the sender, only by the recipient. This model addresses cases for SPAM emails, but it does not work for document dissemination control outside of the intended recipient list.

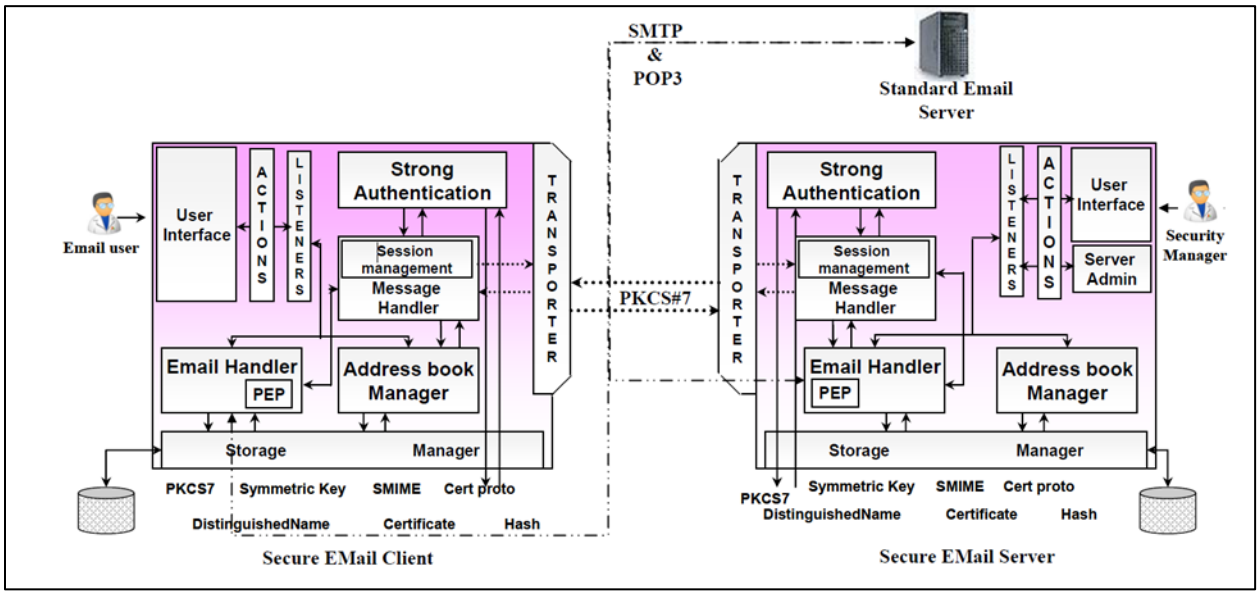


Figure 2.7 - CryptoNET Email Security Model [35]

Mislove et al. [36] proposed a decentralized messaging infrastructure in 2003, called POST, that provides single-copy encrypted message storage in an online peer-to-peer server architecture. This system does not rely upon SMTP for message transmission; instead it relies on peer to peer protocols for message distribution. Additionally, each POST user is required to run a daemon program on their machine that serves as a network node implementing the functionality that would otherwise be provided by an SMTP/IMAP mail server. This daemon program holds

the encryption key for the user, and thus, it presents a per-node vulnerability to the POST ecosystem. However, the main advantage of POST is that a single message is only ever stored once, so system storage is greatly reduced when a message is sent to multiple recipients.

Since the POST system deviates from standard email protocols, all existing email clients are incompatible with POST, and a new worldwide implementation of POST would have to take place in order for the system to provide a viable alternative to standards-based SMTP email. Also, since the POST system requires a peer-to-peer client to be resident on the user's system, mobile device implementations of POST could suffer significant battery life and data bandwidth issues if POST were implemented in those environments. While POST presents the distinct advantage over SMTP in single message encryption, the disadvantages of POST in a large-scale rollout prevent it from being a viable secure email messaging system.

In 2007, Wu et al. introduced a variation of identity-based encryption (IBE) that utilized hardware fingerprint readers to both ensure the identity of the user and to provide a crypto seed source for the encryption of email messages [37]. While this enhancement helped to address concerns about the authenticity of the actual user, it did not provide a mechanism for distributed trust outside of the pre-arranged key trusted parties, nor did it provide trust mechanisms for tiered-based content security or access control.

Each of the above-mentioned systems provides a unique set of ways to handle the shortcomings that are inherent to base-level email. These systems also have their distinct set of disadvantages. As noted by both Wu and Kapadia in 2007 and Chen & Ma in 2008, more sophisticated non-repudiation measures need to be created, and more fine-grained user authentication controls need to be in place for the increased adoption rates of the current secure email systems to take place. Systems such as PolicyMaker, POST, and ABUSE suffer from base-level design issues in

that their implementations, while valid at a smaller scale, do not scale adequately worldwide while maintaining interoperability with existing email systems. CryptoNET's use of a hierarchical PKI-based system provides an excellent baseline system design, but the distributed key architecture creates numerous system vulnerability points. A truly robust and scalable email system is needed to serve as a next-generation communication backbone for government, industry, and individuals who look to have fine-grained control over sensitive content within their email messages. Existing government requirements outlined in the following sections provide the baseline structural requirements of such a system.

MailTrust Introduction

Not only can information be inadvertently sent to users on non-secure systems/servers, there are also no control mechanisms in place to ensure that content accessible to users is controlled in accordance with the parameters specified in the document. If a person were no longer employed by an agency or had their security clearance revoked, they could still receive emails with sensitive content and they are on the honor system that they will not read past the markings in the message. When dealing with matters of national security, relying upon such an honor system is an outdated approach, especially when technology control measures are available that can alleviate many of the aforementioned concerns.

To that end, in the following chapters, we propose that the MailTrust system will address a number of shortcoming surrounding secure email. Existing PKI email systems fail in the following areas:

- Content can be purposefully or inadvertently sent to users without adequate authority to view the content;
- Content can be forwarded on to other non-secure systems;

- Encryption of content is troublesome with the requirement of an a priori key exchange;
- PKI does not allow for message revocation or altering;
- Creating large-scale PKI systems presents scalability concerns; and
- Cross-system interaction through bridges creates centralize vulnerability points.

MailTrust is built upon existing technologies and merges together the requirements specified in DoD 5200.01 Volume 2 to provide a viable replacement for secure email messaging. MailTrust provides unique security measures for situations where both parties require assurance of both the identity and the qualifications of the recipients on both ends of a communication. Specifically, MailTrust can be summarized via the following system-level requirements:

- **Trusted Identity:** Verification of the sender's and recipient's identity via standards-based next generation digital online identity management.
- **Security & Control:** End-to-end secure email solution with message revocation and cancellation at any point in the communication process.

The MailTrust system requires minimal infrastructure changes to realize the benefits of a modern secure email system that can control the security and dissemination of emails as well as reducing the overall email storage requirements system-wide. With MailTrust, assurances across industry and government can be attained, and the infrastructure is fully extensible to private-level individual level messaging. In the following chapter, a detailed breakdown of the content control and trusted identity measures that the MailTrust architecture employs is described in detail. MailTrust leverages a multi-faceted approach to email security that builds upon new best-practice technologies and provides a scalable platform on which new capabilities can be added in the future.

CHAPTER 3 - TRUSTED IDENTITY & CONTENT CONTROL OF MAILTRUST

In this dissertation, we introduce a new secure email architecture called MailTrust. Two of the core pillars upon which MailTrust is based are the need for individuals on both ends of an email communication to have appropriate assurance measures in place to validate the identity of the sender/recipient and the need to control the message content so that information is maintained within the scope of the intended audience. In this section, we present the unique control measures implemented within MailTrust that facilitate the trusted identity management across the entire email security spectrum. With MailTrust, fine-grained control measures are available for every aspect of content control within the email ecosystem.

Trustmarks

The MailTrust architecture is built upon a next-generation online identity management methodology known as Trustmarks. A Trustmark is a machine readable attestation to the possessor's conformance to a defined set of interoperability requirements and it is analogous to a PKI certificate [38], [39]. A Trustmark represents a componentized representation of a unit of requirements and commonly relates to the security, privacy, data and other policies of an organization. Trustmarks are issued in a similar manner as PKI certificates, but they represent units of conformance that allow interoperability of systems. Currently, Trustmarks are being evaluated in a National Institute of Standards and Technology (NIST) pilot under the National Strategy for Trusted Identities in Cyberspace (NSTIC) program. If adopted nationally, Trustmarks could be a foundational component of all future federated identity ecosystems.

Trustmarks are abstracted representations of a unit of system functionality. The implemented manifestation of a Trustmark principle is in the form of a PKI certificate that represents a 3rd party certification of an entity's conformance to the attestation represented by the Trustmark. An example would be an organization's password security policy. A Trustmark could be issued that certifies that an organization enforces a password policy requiring passwords that are 8 characters minimum length, must contain upper, lower, alpha-numeric and special characters, and expire every 45 days. Once this Trustmark is issued to an organization they can use that certificate to cryptographically sign statements about their users that can be leveraged in a federated environment. If another system requires that same policy, then that partner system can allow certain users into their environment that present assertions containing the required Trustmarks.

Trustmarks to date have been used only in the context of Federated Identity Management (FIdM), primarily for the purpose of user-to-system level interactions. Trustmarks go well beyond establishing trusted single sign-on between systems. Since system-level policy attributes can be represented by Trustmarks, a methodology is created whereby a system can evaluate attestations of another remote system's policies. Suppose system A has a minimum password length of 12 but system B, which wants to access system A's resources, only has a minimum length of 8. Those policies can be represented by Trustmarks and evaluated at the machine level via policy decision points (PDP) and policy enforcement points (PEP) that evaluate and enforce business logic transmitted through Trustmarks.

In order to describe how Trustmarks are leveraged in the MailTrust architecture, it is best to look at how Trustmarks can help secure a sensitive government email communication, as illustrated in *Figure 3.1*. In a Trustmark-enabled MailTrust ecosystem, if a president were to email a secretary, the president would first create an email using their preferred email client that

has a MailTrust plugin. The president would note the security level in conformance with DoD 5200.01 standards that specify the appropriate security levels. The secure content would be extracted by the MailTrust plugin and routed to the MailTrust server. The remaining email with a tag reference to the secure content would be sent on to the president's email server and routed to the email server used by the secretary. Upon receipt of the email, the secretary's MailTrust-enabled email client would recognize the document tags and request the secure content from the president's MailTrust server. The request to the MailTrust server would contain a token that identifies the requesting user. The MailTrust server would then authenticate this token against the trusted identity provider that the secretary uses. The secretary's identity provider would respond back with a cryptographically signed XML assertion as to the secretary's user attributes. If the president's MailTrust server validates the secretary's identity provider and the attributes provided by the secretary's identity assertion that is signed with Trustmarks issued to the secretary's identity provider, then the secretary's MailTrust plugin will be allowed to view the secured content. If the MailTrust server is not able to validate the recipient's identity, then no content will be viewable by the recipient.

Providing verification of identity, user attributes, and the associated identity provider creates a granular control framework through which many levels of security may be implemented. The MailTrust architecture has a straightforward data flow that provides an end-to-end security paradigm. Extensive system infiltration attack vectors are analyzed to ensure appropriate security countermeasures exist at each point within the architecture. The use of Trustmarks in this domain is new and innovative, providing a new frontier for secure email access controls.

Trustmarks also enable 3rd party certification of the conformance of a party to a given set of Trustmark assertions. In a manner similar to the way that certificate authorities evaluate businesses before issuing higher level SSL certificates, Trustmark authorities evaluate the security and policies of an organization before issuing the organization the requested Trustmarks. Existing implementations of Trustmarks have focused on user-to-system and system-to-system level interactions, primarily in a web application context. MailTrust is the first architecture to extend this concept into securing email and ensuring non-repudiation.

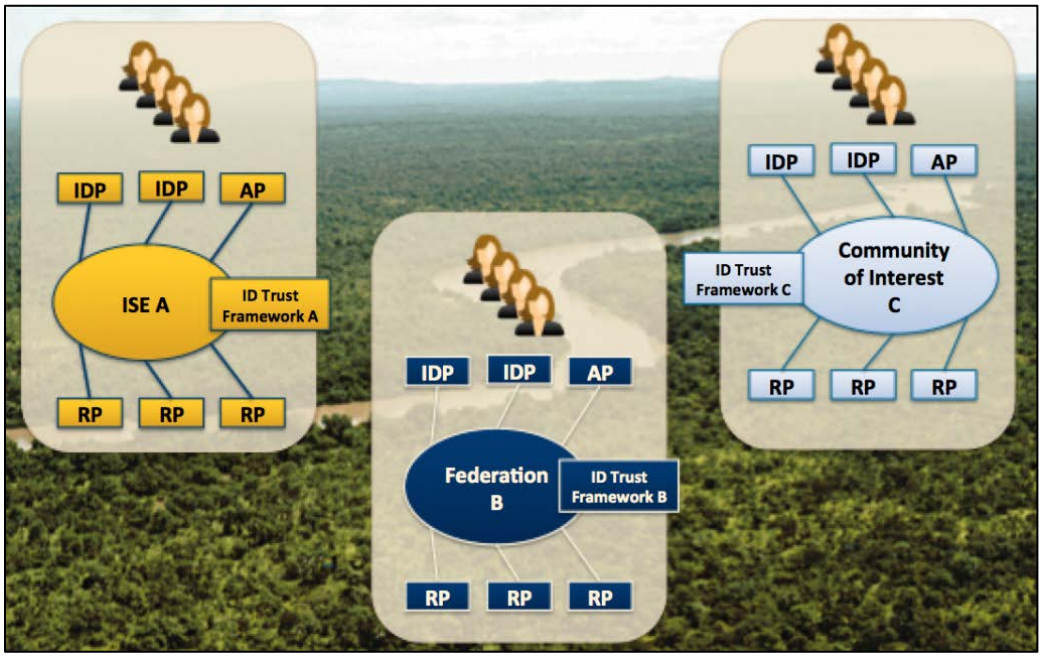


Figure 3.1 - Multiple Isolated Silos of Federations [40]

Web applications are typically silos of application functionality, i.e., data and user repositories meant to service a given business need. Trustmarks were developed to solve interoperability problems whereby multiple identity federations were being created to address application in-

teroperability, but those federations themselves could not interact. The problem with the interaction between these federations was that each of them had its own policies and requirements, and that created islands of monolithic and isolated federations (*Figure 3.1*).

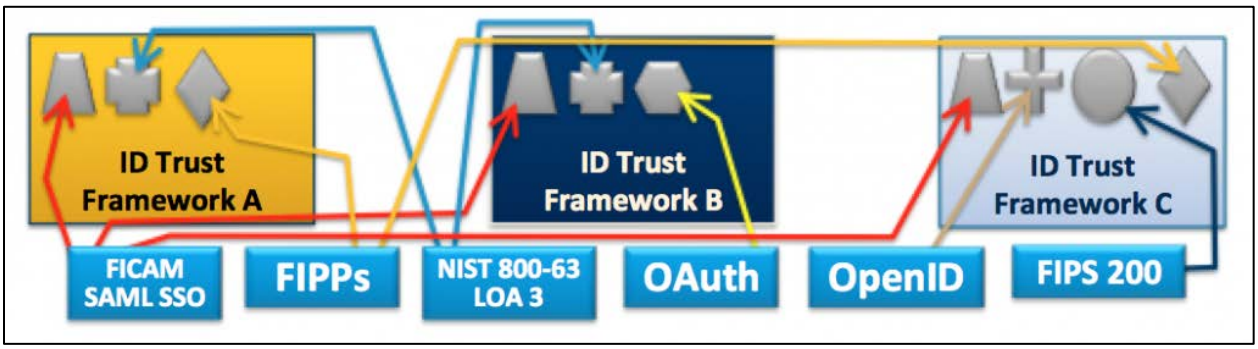


Figure 3.2 - Componentized Trustmark-Enabled Federations [38]

Trustmarks leverage the concepts of PKI and the creation of a componentized policy, security and other business practices so that federations can programmatically interact (*Figure 3.2*). This methodology can be naturally extended to address email system interoperability concerns. Trustmarks and the user assertions that are represented by the cryptographic assertions signed by an identity server can be evaluated in an email context. This evaluation ensures that a users are who they claim to be (non-repudiation), and it also ensures that they have the appropriate authority to view the email content.

Trustmark-enabled user assertions can take multiple forms but are commonly implemented via Security Assertion Markup Language 2.0 (SAML 2.0) or JSON Web Token (JWT). A Trustmark-enabled infrastructure can service many use case scenarios, but the most prevalent at this point in time is a user-to-system case where a user assertion is presented to the system, or service provider (SP). The SP validates the identity of the user by examining the signing Trustmarks presented in the GFIPM SAML assertion. If the identity provider in the assertion passes the requirements set forth in the SP, then the identity of the user in the assertion is trusted to be

valid. The SP then examines the attributes in the assertion related to the user. If the user has the attributes needed to access the resource provided by the SP (often a web page), then the SP grants the user access to the resource.

Trustmarks are encompassed in a greater concept called a Trustmark framework. As illustrated in *Figure 3.3*, this framework consists of a Trustmark spec, a Trustmark Definition (TD) spec and a Trust Interoperability Profile (TIP). These specs are manifested in three real-world instances: Trustmark Instances, TD Instances, and TIP Instances. The Trustmark Spec handles the basic structure of the Trustmark. The TD Instance is a formal definition of the conformance and assessment criteria for each instance of a Trustmark. The TIP Instances use the TD Instances to enforce interoperability concerns between systems.

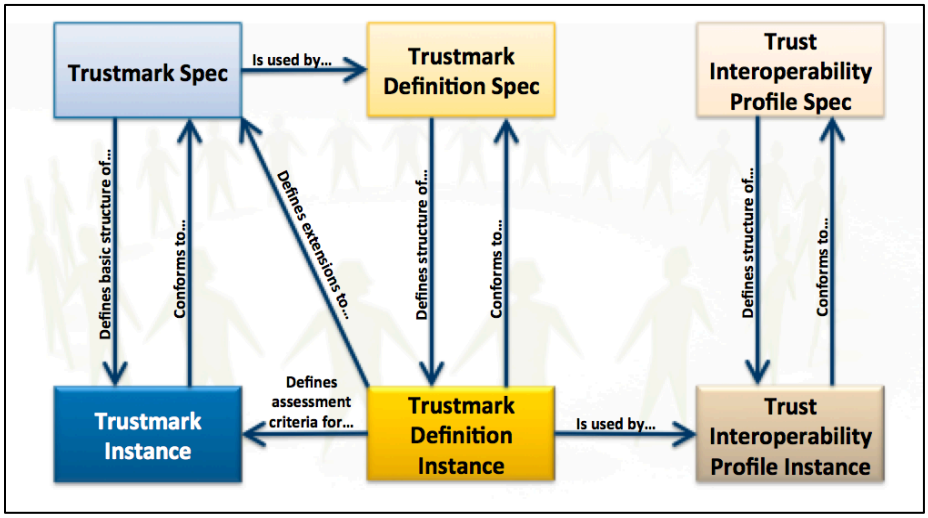


Figure 3.3 - Trustmark Framework Interactions [41]

A Trustmark Definition defines the structure of a Trustmark and ensures consistency, reliability, reuse and usability of the Trustmark documents. Every TD requires the components as illustrated in Table 3.1.

Table 3.1 - Components of a Trustmark Definition

Target	Purpose
Who created the TD?	***Name of the Publishing TDO***

Where is it available	***Canonical Location***
What is it called?	***TD Name***
What is it for?	***Description and Intended Purpose***
Who is it for?	***Target Stakeholder Audience***
Date of Publication	***The Date the TD is Valid From***
Version Number	***Incremental Version to Facilitate Compatibility***

Instances of Trustmark Definitions are required to implement the rules defined in the Trustmark Definition Specification (TD Spec). These rules ensure that the conformance criteria are met, define the process for assessing Trustmark conformance, define the criteria for Trustmark certification, and provide additional metadata about the TD such as the publishers, URL, etc.

Information Assurance Through Trustmarks

As an example of how MailTrust can leverage the framework of Trustmarks, we consider FIPS (Federal Information Processing Standards Publication) Publication 199. This publication mandates that agencies provide security categorizations for their information systems as low-impact, moderate-impact, or high-impact for the security objectives of confidentiality, integrity, and availability. An impact assessment value is calculated that corresponds to the security objectives based on the highest security mark across systems. We begin by calculating a security category (SC) using the following set function:

$$\mathbf{SC}_{\text{information system}} = \{(\mathbf{confidentiality}, \textit{impact}), (\mathbf{integrity}, \textit{impact}), (\mathbf{availability}, \textit{impact})\},$$

Figure 3.5 – FIPS Security Category Set Function

The valid values for potential impact are low, moderate or high. As stated in FIPS Pub 199:

- For *low-impact* information systems, organizations must, as a minimum, employ appropriately tailored security controls from the low baseline of security controls defined in NIST Special Publication 800-53 and must ensure that the minimum assurance requirements associated with the low baseline are satisfied.

- For *moderate-impact* information systems, organizations must, as a minimum, employ appropriately tailored security controls from the moderate baseline of security controls defined in NIST Special Publication 800-53, and they must also ensure that the minimum assurance requirements associated with the moderate baseline are satisfied.
- For *high-impact* information systems, organizations must, as a minimum, employ appropriately tailored security controls from the high baseline of security controls defined in NIST Special Publication 800-53, and they must also ensure that the minimum assurance requirements associated with the high baseline are satisfied.

MailTrust incorporates the mandates defined in FIPS Pub 199 for information security assurance through the leveraging of Trustmarks. Trustmarks enable security control extensions well beyond that of which traditional email messaging provides. Whereas DoD manual number 5200.01 Volume 2 specifies the policies and procedures for marking of classified information, a gap between FIPS Pub 199 and DoD 5200.01 exists. Specifically, 5200.01 addresses security requirements but does not provide defined methodologies for ensuring the implementation of those requirements within the context of emails. MailTrust bridges this gap by leveraging Trustmarks to provide information assurance (IA) controls to sensitive data throughout the information lifecycle. Information assurance is the managing of information related risk concerns. Each MailTrust server acts as a policy enforcement point (PEP) and ensures that the required user attributes are met before any information is ever disseminated to a requesting user. The information assurance ratings defined in Pub 199 require that “high-impact” information, corresponding to Secret/Top-Secret information in DoD 5200.01, be protected with the high baseline controls covered in NIST 800-53. NIST 800-

53 specifically calls for stringent non-repudiation, identity management and information assurance controls in relation to email and user identities.

Non-repudiation Control

The AU-10 non-repudiation control is meant to protect individuals and organizations against false claims of having performed a set of actions, and to prevent the denial by a party for having performed an action. Currently, non-repudiation is handled via a government-wide PKI implementation. Emails are digitally cryptographically signed by certificates that are uniquely assigned to system users, and this signature is verifiable on the recipient's end. However, the PKI non-repudiation model does not address the inappropriate dissemination of information, nor does it address the control mechanisms around the message information. This is where Trustmarks play a key role in the MailTrust architecture.

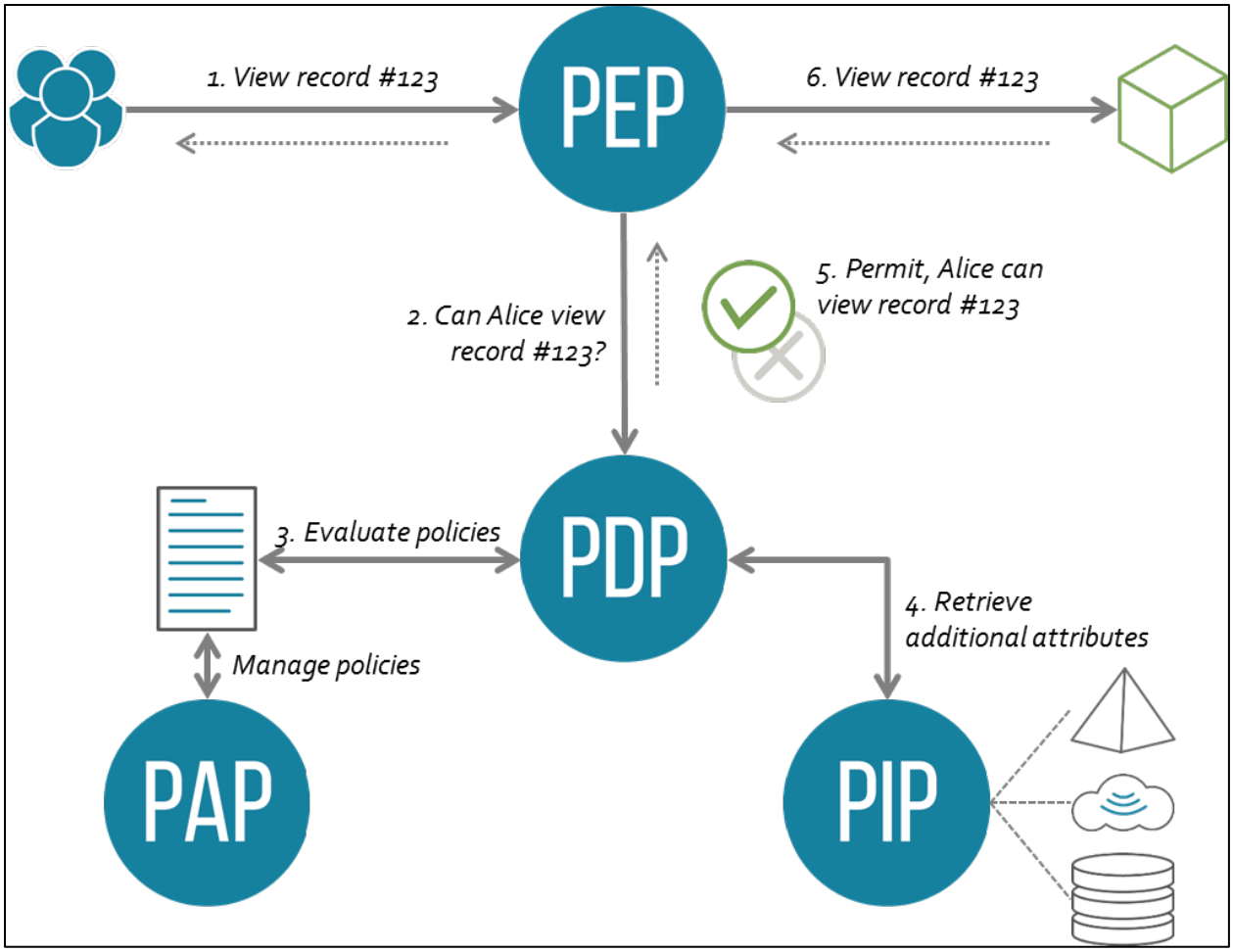


Figure 3.4 - XACML Policy Control Flow [42]

As seen in *Figure 3.4*, if a user wants to retrieve a message, a user assertion is presented to the MailTrust Policy Enforcement Point (PEP). The PEP then forwards the request to the Policy Decision Point (PDP), which evaluates the request in the context of policies that are stored in the Policy Administration Point (PAP). If the PDP determines that the user has authority to view the information, then the PDP approves the PEP, which retrieves the data and returns it to the user. The PDP can also pull in addition information for policy decisions from connected nodes called Policy Information Points (PIP). MailTrust utilizes this framework to provide the multi-tier evaluation of requests from users. Not only is the identity of the requesting user validated, but any aspect of the user's profile and the user's authentication provider can be used in policy

decisions. This framework also enables partial-redaction capability for documents and can also enable content changing based on user attributes that will be discussed further in Chapter 4.

In connection with non-repudiation comes the ability to bind an identity of an information producer to the information generated. In addition, the method by which authorized personnel are able to process non-repudiation procedures is directly tied into the identity management system.

Identity Management

NIST 800-53 defines the IA-4 Identity Management scope through which federal agencies must implement their IT infrastructure. The scope of these requirements covers:

> *a. Receiving authorization from [organization-defined personnel or roles] to assign an individual, group, role, or device identifier; b. Selecting an identifier that identifies an individual, group, role, or device; c. Assigning the identifier to the intended individual, group, role, or device; d. Preventing reuse of identifiers for [organization-defined time period]; and e. Disabling the identifier after [organization-defined time period of inactivity].*

These baseline controls are further defined by the following overarching identity management criteria:

- User registrations that result in an individual identifier must be conducted in person, and a supervisor must approve the identity.
- During registration, multiple forms of certification of individual identification, such as documentary evidence or a combination of documents and biometrics, must be presented.
- Identities must be unique and each person must have their own singular identity.

- Identifiers must be managed dynamically.
- It is prohibited to use the same identifier for a user account that accesses an information system for that of a public identifier for individual electronic mail accounts.
- Coordination with external organizations takes place through cross-organization management identifiers. Cross-organization identifier management provides the capability for organizations to appropriately identify individuals, groups, roles, or devices when conducting cross-organization activities involving the processing, storage, or transmission of information.

To implement the aforementioned FIPS 199 and NIST 800-53 control criteria, MailTrust leverages the Trustmark concepts to address the following set of digital control concerns:

- Information assurance
- Non-repudiation
- Unique identities / Separate login from email
- Dynamic identity management
- Identity coordination with external organizations

Each of the above control concerns is addressed within the MailTrust architecture in the following sections.

MailTrust Information Assurance

DoD defines these information security categories as Unclassified, Confidential, Secret and Top Secret while FIPS 199 defines their categories of IA as low, moderate and high impact. Regardless of the classification methodology used, an architecture needs to be in place that will facilitate the attribution of data and security measures that ensure the data are only viewed by persons with the appropriate authority.

Trustmarks are built on the concepts of Attribute Based Authentication (ABAC), whereby specific user attributes are asserted, and a target system evaluates these attributes to determine what access levels that the user will be able to attain [43], [44]. Basic email has no inherent document security, and S/MIME only provides encryption to specified target users that are within the realm of the PKI infrastructure between two users. Also, S/MIME only has control measures to validate that a given user is who they claim to be; it does not have controls in place to verify whether or not the target recipient has the corresponding attributes to view the security level of the content that is being sent to them. The following is a base example of where both regular email and S/MIME break down.

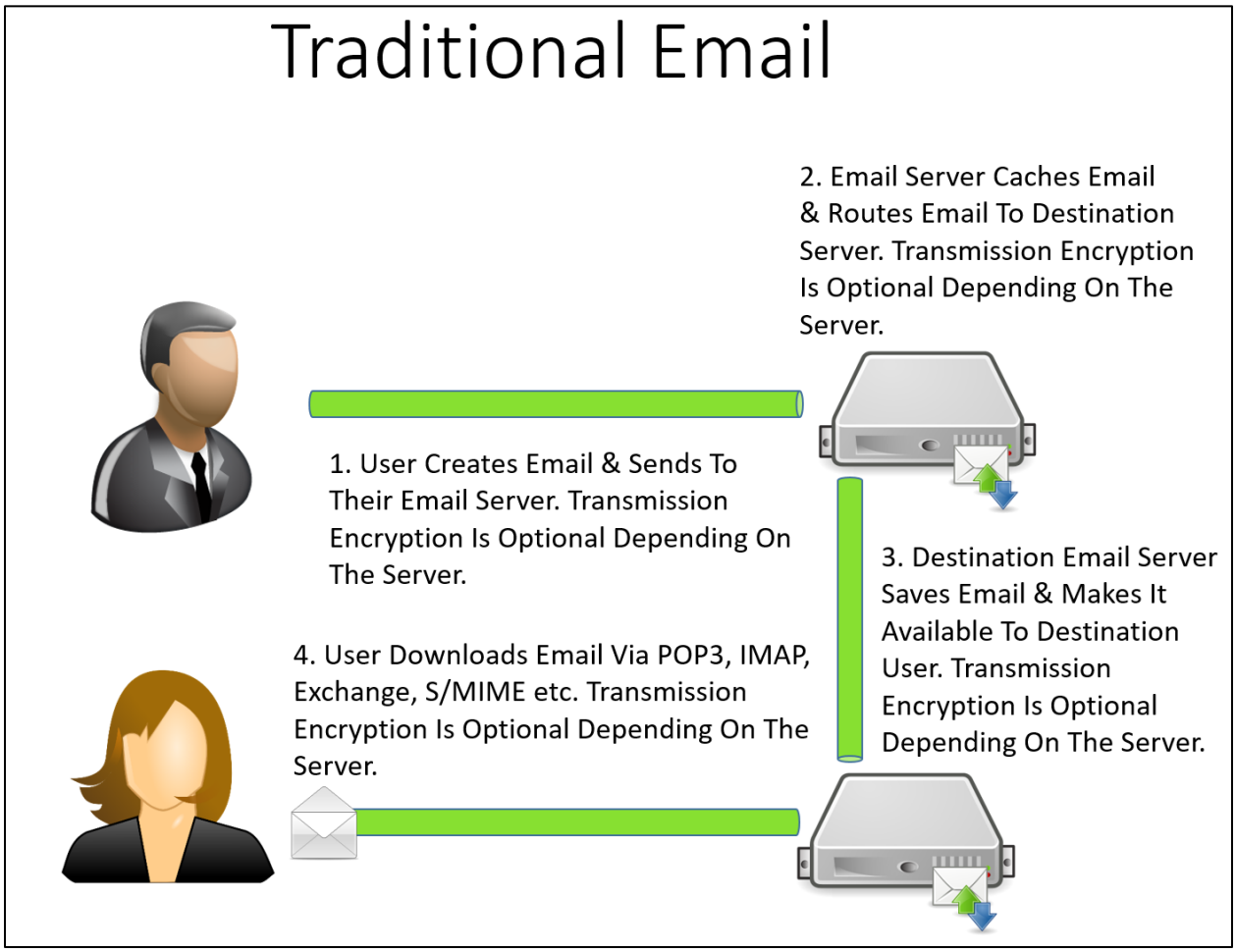


Figure 3.5 – Traditional Email Data Flow

Regular Email

As seen in *Figure 3.5*, in a traditional email data flow model, multiple copies of email exist across many systems, and these data transmission may not be protected at each level. To illustrate this data flow, consider the following example: Bob wants to email Alice. Bob types in Alice's email address and sends her a message through his email client. Through traditional email, this message is not secured and could potentially be viewed during transmission, on intermittent email relay servers, or by a person who hacks into or steals Bob or Alice's machine. This method of communication should be viewed as public in nature as multiple points along the communication pathway are vulnerable. Classified or greater information, or information that is moderate impact or higher, should never be transmitted in this manner.

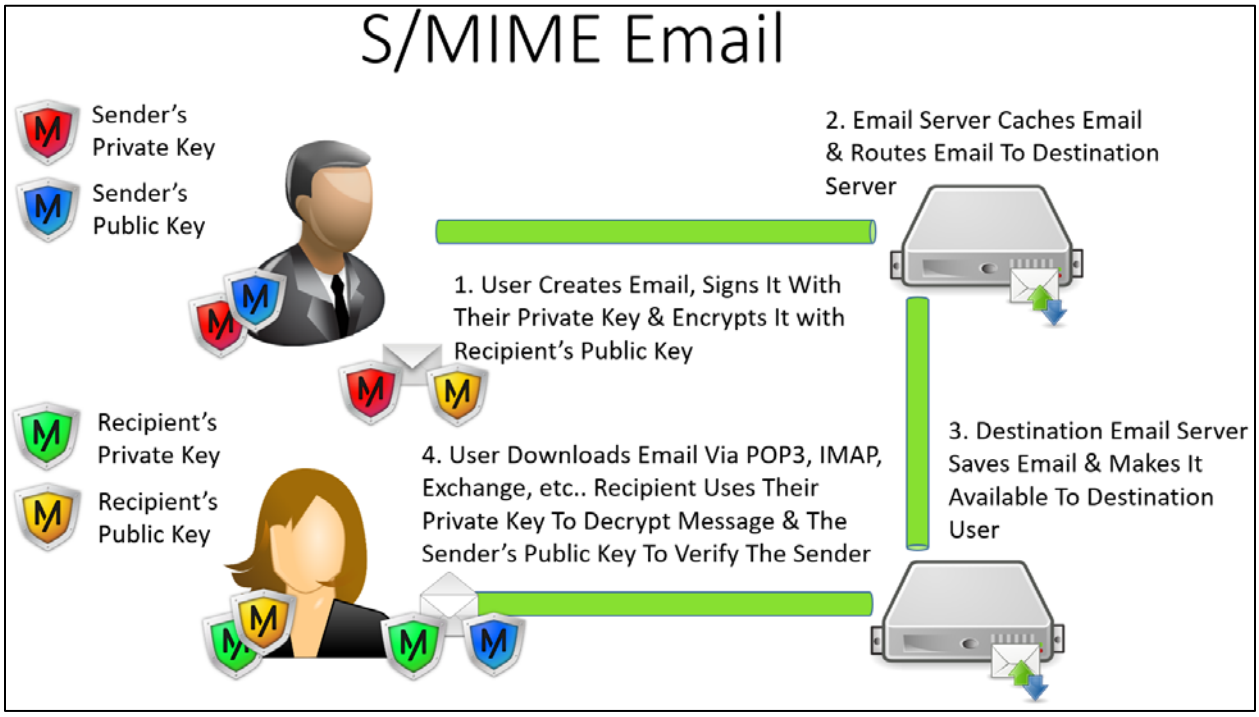


Figure 3.6 – S/MIME Email Data Flow

S/MIME Secure Email

Figure 3.6 diagrams the data flow for an email sent using the S/MIME protocol. In the first step, a user authors an email and then signs the email with their private key. The user then encrypts the email content with the recipient user's public key. After transmission and download, the recipient can then decrypt the email using their private key due to the asymmetric properties of the PKI key pair. The user can then verify the authenticity of the sender using the sender's public key. To illustrate this flow, an example is given below.

Bob wants to email Alice. Bob works for the Army and Alice was a contractor who worked for the Department of Homeland Security. Bob drafts an email that contains sensitive information and sends the email securely using the S/MIME protocol. The message will only be viewable by Alice and will be protected in transit. However, problems begin when Alice receives the message. Alice has taken another job in the private sector but still has her DHS email account. Alice also has never completed her mandatory security training. Not only does Alice not have the required training to view the sensitive content, but she no longer is affiliated with her prior agency, so her clearance to view the content has been revoked. S/MIME does not protect against the cases where the user does not have the right content, nor does it adequately protect against the forwarding of secure content to unauthorized individuals. S/MIME is a major step forward from basic email, but there are significant hurdles that still must be overcome for the system to cover all of the operational goals defined in both FIPS 199, NIST 800-53 and DoD 5200.01.

MailTrust Features

MailTrust vastly differs from S/MIME in that secured content never travels through email servers, and viewing of the content is based on dynamically determined user attributes. MailTrust extracts out the secured content and stores that content on the sending user's email server.

The content of the email is then replaced with a GUID (Globally Unique Identifier) reference to the stored content. Below is an example of a message sent through MailTrust:

Pre-MailTrust:
To: Alice
From: Bob
Subject: Project Help
Body: Alice, I hope you are doing well. The report back from the field that I received today indicates that insurgents have moved past the riverfront and are making their way through the countryside. We need your team to extract out the latest satellite intel and correlate the imagery with SAT-CON 3 thermal views to give the ground force the information they need to determine the appropriate battle plan.

Post-MailTrust:
To: Alice
From: Bob
Subject: Project Help
Body: <MailTrust>06068bf4-0333-40b9-854b-47e800c8fc89</MailTrust>

Figure 3.7 – MailTrust Body-Only Example

The extraction of the content allows the message to be transmitted via SSL transport-level encryption, or even encrypted email, since the content has been removed. Once a recipient receives the message, the MailTrust-enabled email client on the recipient's machine reads the email and parses out the MailTrust tagged items. The user's client then contacts the sender's server to view the sensitive information. Attributes about the recipient user account are shared with the sender's server via Trustmarks, and the sender's server then evaluates those assertions to decide whether or not to render back the requested content. Since the content is stored securely on the sender's server and only ever transmitted to approved MailTrust-enabled clients, information assurance is maintained along the entire pathway.

There are times where the email subject itself may be sensitive and the MailTrust architecture can easily support subject–level content encryption in the same manner as the body content:

Pre-MailTrust:

To: Alice
From: Bob
Subject: Project Help
Body: Alice, I hope you are doing well. The report back from the field that I received today indicates that insurgents have moved past the riverfront and are making their way through the countryside. We need your team to extract out the latest satellite intel and correlate the imagery with SAT-CON 3 thermal views to give the ground force the information they need to determine the appropriate battle plan.

Post-MailTrust:

To: Alice
From: Bob
Subject: <MailTrust>ad7552b5-f98c-4106-b1f1-6b8fe3ae5b37</MailTrust>
Body: <MailTrust>06068bf4-0333-40b9-854b-47e800c8fc89</MailTrust>

Figure 3.8 – MailTrust Subject & Body Example

As seen in 3.9, both the email body and the subject are MailTrust protected. The recipient's email client with MailTrust-enabled functionality would receive the email, observe the MailTrust GUID, and then retrieve the content for both the subject and the email body from the sending user's MailTrust server.

By encrypting the entire body of the email rather than just the sensitive content, workload is reduced on the user, the sending message size is smaller, and the possibility of data-leakage due to "covert channels" is reduced [45]. The existence of encrypted content in a contextual basis would inherently create a partial data leakage scenario, since a user could potentially infer the existence of secured content based on the placeholder.

Insecure data protection model:

To: Alice

From: Bob

Subject: Project Help

Body: Alice, I hope you are doing well. The report back from the field that I received today <MailTrust>06068bf4-0333-40b9-854b-47e800c8fc89</MailTrust>. We need your team to extract out the latest satellite intel and correlate the imagery with SAT-CON 3 thermal views to give the ground force the information they need to determine the appropriate battle plan.

Figure 3.9 – Insecure Method of Email Content Protection

While the example in *Figure 3.9* removes the sensitive content, it leaves behind other information that is confidential within itself, since by its proximity to secure information, the viewer can inherently gain proximal knowledge of secure content. Due to this possibility of cover channel data leakage, the MailTrust architecture adopted a complete replacement model.

MailTrust Non-repudiation

Non-repudiation, or the validation that a document was authored by a specific individual, is achieved through the cryptographic signing of the secure content that is stored on the MailTrust server. When content is tagged in a MailTrust email, the content is extracted and encrypted on the MailTrust server using a key that corresponds to the user sending the message. Without proper authentication from the user, the MailTrust server cannot attain that encryption key needed to secure the associated content. This encryption with an authentication byproduct (user JWT token key) creates an irrefutable tie between the author of the message, the identity provider, and the content of the message.

Validation of the recipient's identity is also a key pillar of non-repudiation in the MailTrust architecture. A recipient is not able to view secured content until the user presents a valid token. This token is then verified against the user's identity provider which in turn is validated

via Trustmarks that were issued to that identity provider. This end-to-end validation of both sender and recipient allows MailTrust to provide a degree of non-repudiation that is much higher that many other secure email ecosystems.

MailTrust Unique Identifiers / Separate login from email

MailTrust maintains the control requirement for unique identifiers and for a login separate from a user's email address by enforcing the security constraints that are stored within the affiliated user repository. Assertions as to the policy compliance of a repository are a pillar of how Trustmarks are implemented. A Trustmark that represents the conformance to a set of security policies would be issued to each identity provider (IDP). As content is requested from a MailTrust server, the assertions of the incoming user are evaluated by the MailTrust server. If a user receives an email from a MailTrust user but the receiver's account is associated with an identity server that has not met the security certifications via the corresponding set of Trustmarks, then that user would not be able to view that sensitive content since their login account did not meet the required criteria.

MailTrust Dynamic identity management

The required control to be able to dynamically manage users fits in perfectly with the concept of Trustmarks. Every transaction is evaluated on a per-transaction basis. If a user's privileges are revoked or elevated, then the MailTrust server will, on the fly, render back the content that the affiliated user is allowed to see. A user's assertion of identity is contained within a JavaScript Object Notation (JSON) or XML-based Security Assertion Markup Language (SAML) packet that is transmitted to the MailTrust server when the recipient requests a secured segment of a document. A request to an IDP for an authentication token can take place via a POST of data to an SSL protected RESTful-based service in the form show in *Figure 3.10*.

Token request validation success


```
{
 "ClientId": "ro.client",
 "ClientName": "Resource Owner Flow Client",
 "GrantType": "password",
 "Scopes": "email openid roles",
 "UserName": "trustmarktest@gmail.com",
 "AuthenticationContextReferenceClasses": [],
 "Raw": {
  "grant_type": "password",
  "username": "trustmarktest@gmail.com",
  "password": "******",
  "scope": "roles openid email"}
}
```


Figure 3.10 – MailTrust Token Validation Request

Once a token is obtained from a user's IDP, then that token, in the form of a JSON Web Token (JWT), may be used to assert identity to a MailTrust server. The user then requests the secure content by posting the token to a MailTrust server service as shown in *Figure 3.11*.

Token

"MailTrust":" 06068bf4-0333-40b9-854b-47e800c8fc89"
"JWT":
"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2p3dC1pZHAuZXhhbXBsZS5jb20iLCJzdWIiOiJtYWlsdG86bWlrZUBleGFtcGxlLmNvbSIsIm5iZiI6MTQ3NTE4NDI2OSwiZXhwIjoxNDc1MTg3ODY5LCJpYXQiOjE0NzUxODQyNjksImp0aSI6ImlkMTIzNDU2IiwidHlwIjoiaHR0cHM6Ly9leGFtcGxlLmNvbS9yZWdpc3RlciJ9.yb2OTYYdgHGySN3rR62OlBGYu0m8ro-lHbhMkn7_g9qfaN_SjFhD4gUTFeF9RwZdrhh2SUTpsIhgskIOZyNgvQ"

Figure 3.11 – Example MailTrust Token

The MailTrust server validates the JSON Web Token and sends a request to the IDP specified in the JWT with the corresponding user account. The IDP responds back with an assertion of the identity of the user coupled with the cryptographic signing of the identity assertion using the Trustmarks that have been issued to that identity provider.

Figure 3.12 – Example MailTrust Validated JSON Web Token

The identity server's assertion of a user's attributes is validated by the MailTrust server via verification of the Trustmarks that were issued to the user's IDP. An example of the JSON web token validation can be viewed in *Figure 3.12*. Through the PKI equivalent of a Root Certificate Authority chain, a Trustmark chain establishes a verifiable trust path between two disparate servers.

MailTrust Identity coordination with external organizations

One of the main advantages of MailTrust above other secure mail solutions is the scalability and integration potential of the platform. The current DoD PKI landscape is a tightly wo-

ven architecture that relies solely on bridge server systems to allow external organizations to validate users. Alternatively, the Trustmarks on which MailTrust is based allows a dynamic federation of entities. The federation can be based on explicit relationships such as "The Department of Defense trusts users from The Department of Homeland security" or on implicit relationships as in "The Department of Defense trusts users who have Trustmarks that have Secret or Top Secret attributes." Explicit relationships are hard to manage at large scale, so having another route for coordination that is based on implicit relationships becomes essential as the network grows to thousands or millions of nodes.

The POST secure email system proposed by Mislove et al. in 2003 contained some of the baseline principles adopted by the MailTrust architecture, but the use of Trustmarks coupled with utilizing aspects of existing SMTP infrastructure for base level message transmission provide the key differences that enable the MailTrust architecture to attain a faster adoption/integration rate while maintaining scalability. In Chapter 6, the overhead associated with message encryption is examined through a series of simulations. The aggregate performance results indicate that MailTrust is a scalable architecture that is capable of growing to meet the internet's expanding security needs.

By leveraging Trustmarks, MailTrust is able to implement IA measures such as dynamic identity management, non-repudiation controls, and multi-organizational coordination. MailTrust extends Trustmarks to this new security domain and addresses the shortcomings of existing and proposed technologies. In the next chapter, the security and control measures of MailTrust are examined in depth. A prototype MailTrust system is discussed and analyzed to show how the MailTrust architecture can be implemented in a real-world environment.

CHAPTER 4 - SECURITY & CONTROL OF MAILTRUST

MailTrust maintains end-to-end security through the access control measures inherent to the system design, coupled with the use of Trustmarks to validate user authority, all in conjunction with the hybrid message storage and transmission architecture. Since non-sensitive message data is transmitted through existing SMTP/S/MIME routes, the MailTrust architecture maintains interoperability with all existing email systems. Plugins for clients only need to be developed to handle the content that is protected by MailTrust. Through this hybrid secure/insecure approach, MailTrust could be adopted and integrated into email solutions in a piecewise manner. MailTrust provides rigorous encryption with the addition of user attribute assertion, and even allows the user to revoke messages as well as specifying security levels.

Message Revocation & Tracking

One of the unique aspects of MailTrust is the ability for a user to cancel or revoke a portion of the secured content of a message. Once a message has been sent, the identifier of the content is embedded in the message as a reference handle. At any point in time a user has the ability to edit sent content since the core message still resides on the user's server. The user can even remove all of the referenced content, thereby effectively implementing a message revocation capability. This dynamic message-altering and revocation capability has existed in online social media communications for years, but this capability has never existed in the context of a secure email system.

As part of this dissertation project, other control measures have also be researched. Since MailTrust centrally stores the secured content, verification measures can be put in place to log

(1) when recipients view messages, and (2) from what client and location using IP information. That log can then be made available to the sender to give insight into whether or not users have viewed a particular message.

Document Security

The MailTrust architecture encrypts each stored message with 256bit Advance Encryption Standard (AES) encryption [46]. The AES encryption utilizes a 256bit random salt hash coupled with a 256bit random entropy generated initialization vector (IV). A CBC cipher mode and PKCS7 padding are used for the symmetric key. The salt value is iterated 100,000 times over when generating the salt bytes from the RFC2898 compliant salt function utilized in MailTrust. The corresponding AES passphrase is based on a token value for the sending user that is provided by the user's identity provider. Since the message decryption components rely on an external server component for decryption (the sender's IDP), this mitigates any concerns about compromising a MailTrust server. The message, salt, and corresponding user token key, which resides on the IDP server, are all required in order to decrypt the message content. All hashes in MailTrust are calculated with SHA-256 as SHA-1 was deprecated by NIST in 2011 and recently broken by researchers [47]. MailTrust uses SHA-256 hashing in the client to server verifications, see APPENDIX C.1, as well as certificate signing of response messages that is further described later in this section.

Through the use of Trustmarks, a user can specify the document security levels associated with the content of a given message. The base level cases are items like Secret and Top Secret, whereby a recipient must have the corresponding user attributes to view the content within the message. The MailTrust architecture, however, goes well beyond that base use-case model. Global user or enterprise messaging profiles can be established that enforces message security

across all secure transmission. For example, an organization could establish the following system-wide protocol:

"Require all recipients to have a FICAM LOA 3 or higher identity & all recipients MUST be FBI CJIS background checked within the past 6 months"

The above security profile would be enforced at the MailTrust server level of the organization and would require the following assertions from users requesting content from the server:

REQ: FICAM LOA3 IDP

REQ: CJIS_VERIFIED && 6<=DATEDIFF(MONTH, CJIC_ VERIFIED _DATE, NOW())

A user requesting content from a MailTrust server presents their JWT token to the system and the MailTrust server validates that token against the user's IDP. After verification, the MailTrust server requests assertions from the IDP about the attributes required to access the content. If the user has the requested attributes, then the IDP responds back with the values of the attributes. The response is cryptographically signed with the certificates issued to the IDP during the Trustmark certification process of the identity provider. The MailTrust server is then able to verify the assertion of the attributes by verifying the trust chain through the PKI trust established back to the Trustmark root CAs.

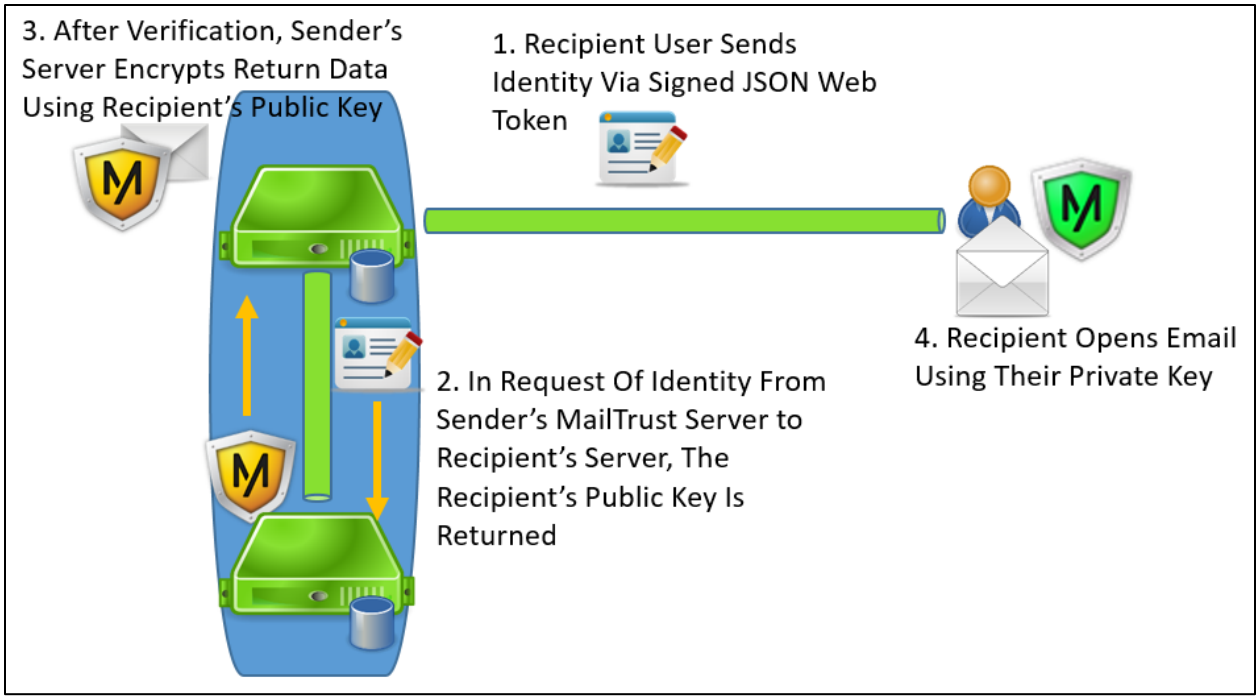


Figure 4.1 – MailTrust Message-Transit Protection

Since the assertions have been validated, the MailTrust systems can now trust their validity, and it returns a decrypted version of the requested content back to the message recipient. In order to protect the message in transit and ensure that only the appropriate person views the message, the MailTrust server utilizes the public key of the recipient. As seen in *Figure 4.1*, the MailTrust server encrypts the response message with the recipient's public key. The public key of the recipient is retrieved from the identity provider associated with the recipient. Since a trust relationship exists between the two identity providers, the sender's server can both validate the JWT of the recipient and retrieve the public key that corresponds to the recipient from the identity provider.

The key exchange between the two identity providers occurs out of band from the recipient user, so a malicious user is not able to influence that communication process. Once the sender's identity provider receives the public key, the provider can then encrypt the response

message with the user's public key. Since data encrypted with a public key can only be decrypted with the corresponding private key of the public/private key pair [48], MailTrust is able to ensure that, beyond just transport level SSL encryption, the user who views the message is indeed the one who has authorization as verified by the data presented in the JWT.

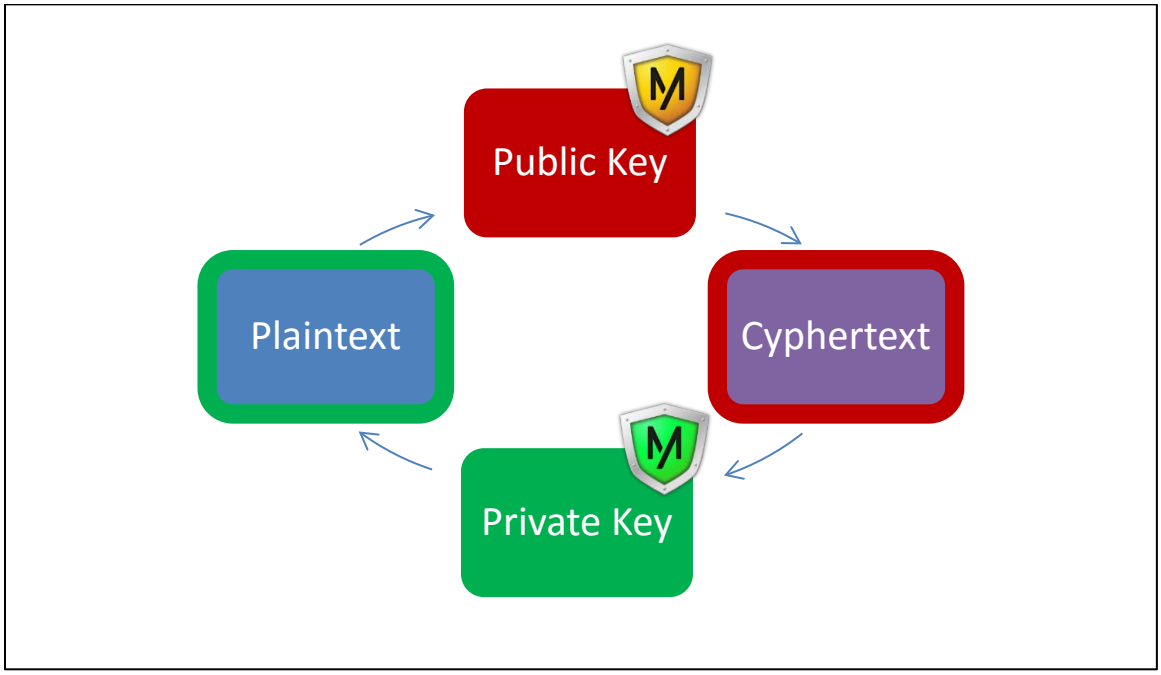


Figure 4.2 – Asymmetric Key Encryption/Decryption

Asymmetric key encryption utilizes a 2 key pair model (public/private keys). This is the foundation of PKI and S/MIME email protection. MailTrust uses the same principles to protect data in transit from the server to the user. The public key encrypts the plaintext with a publically known cypher. The public/private keys are generated simultaneously and are bound and unique to one another. Only the user with the private key that corresponds to the public key will be able to decrypt the information that was encrypted with the public key. *Figure 4.2* gives a visual representation of the flow process flow of the encryption/decryption.

The message revocation and tracking features as well as the overall document security model for MailTrust provide conformance to FIPS Pub 199, DoD 5200.01, and NIST 800-53 requirements for information assurance measures. The ability for messages to be revoked at any point or for security levels to be elevated or reduced after sending are unique to the MailTrust architecture and provide a thorough security model that meets current and future needs. A message only partially exists in current email implementations via opt-in message receipts. In MailTrust, the sender has a full view over every user who has read the content of any MailTrust message. In order to test the security principals proposed in MailTrust, a prototype MailTrust client/server system was created as a test platform to prove the feasibility of the proposed architecture. This prototype system is detailed in Chapter 5 and includes a detailed view of the message protection methodology used in both the MailTrust client and server, as well as a comprehensive overview of the protection measures in place for messages in transit.

CHAPTER 5 - PROTOTYPE MAILTRUST SYSTEM

A prototype MailTrust client/server system was developed as part of this research to prove the viability of a MailTrust implementation as well as to test the performance overhead and gains associated with a MailTrust implementation. In the following sections, development and architecture details about the client and server systems are provided as well as a comprehensive overview of the message flow process involved. This prototype MailTrust implementation resulted in a functional system, but significant development efforts are still needed to create a production-quality MailTrust system.

Example MailTrust Server

The prototype MailTrust client and server were designed using a combination of open source software tools and Microsoft's .NET C# 4.6.1 and was compiled in Visual Studio 2015. A MailTrust server was designed to securely store email content and to process user authentication transactions. A front-end "MailTrust enabled" client was also created to interact with the MailTrust server and to send the email handles through standard SMTP.

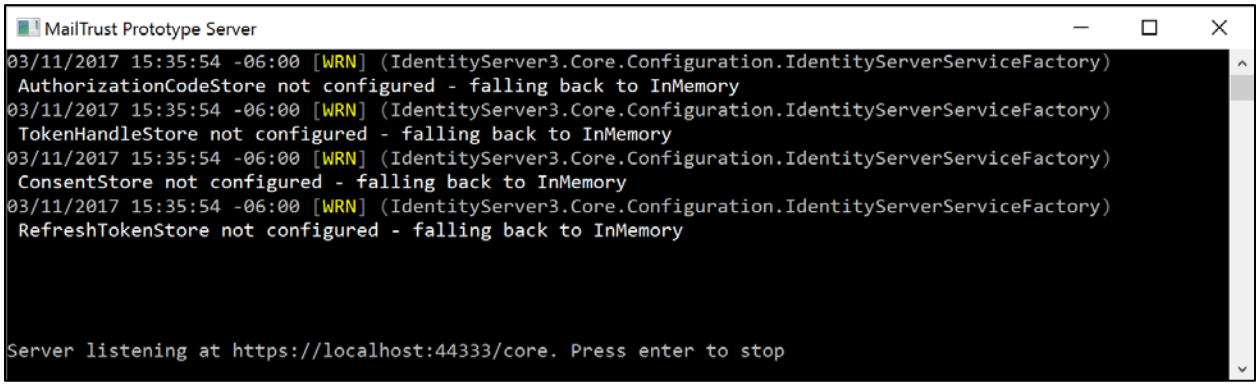


Figure 5.1 – Prototype MailTrust Server

Figure 5.1 shows the startup output of the prototype MailTrust Server. The server, based on IdentityServer3, an open source OpenID Connect and OAuth 2.0 server for .NET, was used in the test implementation of the MailTrust architecture to function as the back-end identity infrastructure [49]. The software functions as both and *Authentication as a Service* platform and as a *Federation Gateway* within the test environment that was setup to examine the MailTrust architecture. IdentityServer3 was modified and extended slightly to support some of the MailTrust specific needs, but fundamentally the software was used almost as-is to handle the trust federation and identity management. For ease of manipulation and debugging during development, the MailTrust server was implemented using a command line host application framework. With minimal changes though, the server could be moved over to a system service or other application hosting container.

The features of IdentityServer3 are highly customizable and configurable. Complete examples of the customization and configurations made can be seen in APPENDIX C & D. APPENDIX A also details the output of the server from startup, to user login, and finally to role identity query and authorization. While IdentityServer3 was used in the test implementation of MailTrust, any identity server platform could be used to implement MailTrust as long as the platform supports the open federation architecture requirements of Trustmarks.

Web methods were added to the identity server host code that facilitated the encryption/decryption and data storage needs of MailTrust. In a production real-world environment, these functions could be divided into separate servers. The encryption methods described in the *Document Security* section above were implemented in this web services layer within the authentication server. The encryption/decryption methods used can be seen in APPENDIX E. The

methods use .NET RijndaelManaged class to perform the encryption and use Rfc2898Derive-Bytes function from System.Security.Cryptography that generated the encryption key using PBKDF2 based off of the string supplied. In the case of MailTrust, that string corresponds to the identifier of the sending user.

Google Gmail was used as the SMTP (email message sending) and as the IMAP (email message sending) platform in the test MailTrust platform that was setup to evaluate this research. Since SMTP & IMAP are standard protocols, the use of Gmail can be easily replaced with any other email server system for the implementation of MailTrust in other scenarios. To maintain transmission security, SSL over port 465 to Gmail's secure server, smtp.gmail.com, was used. Likewise, the IMAP connection was also done over SSL via port 993 to Gmail's secure IMAP server, imap.gmail.com. MailTrust removes the content of the secured emails and provides non-repudiation and various levels of protection, so SSL in the transmission is not required, but SSL on transmission is still recommended to reduce covert channel information data leakage. The existence of any kind of secured transmission of data from one party to another might become the hints that tip off adversaries who can then further target personnel through means other than email, so secured transmission of emails using SSL transport encryption is still recommended.

Example MailTrust Client

MailTrust maintains compatibility with existing email systems given that non-MailTrust users will still be able to receive the handles to MailTrust content. However, they will not be able to view the content if their clients are not MailTrust enabled. By maintaining compatibility with existing email systems for non-secure information, the MailTrust architecture significantly increases the potential for largescale industry and government adoption. In the MailTrust opt-in

model, only those that have a secured MailTrust client and those who have the appropriate credentials will be able to view the centrally-stored secured content. By maintaining control of dissemination and access of secured information, MailTrust solves the gaps that exist in other email ecosystems.

A prototype MailTrust client was created to test out the proposed MailTrust architecture and to have a platform on which performance simulations could be performed. A .NET Winforms application was created for the MailTrust client and it is responsible for the user interface for login to the identify provider, email authoring, and email transmission. Web service calls were made to the back-end identity provider to authenticate users via a simple username and password dialog. Since MailTrust is identity provider independent, multi-factor authenticate (MFA) and multi-step authentication (MSA) are also inherently supported, depending on the capabilities of the identity provider.

Figure 5.2 – Prototype MailTrust Client

Figure 5.2 shows the MailTrust client immediately after login. This user interface lists out message handlers that have been downloaded from the Gmail server. The sender's name and email are visible to the recipient as well as the message download date and time. The subject is

also optionally visible, depending on if the sender indicated that the subject should also be Mail-Trust protected. In the MailTrust prototype client, double clicking on a message line would open up a particular message and allow for the user to retrieve the message content from the MailTrust server. The File menu allows a user to create and send new MailTrust protected emails.

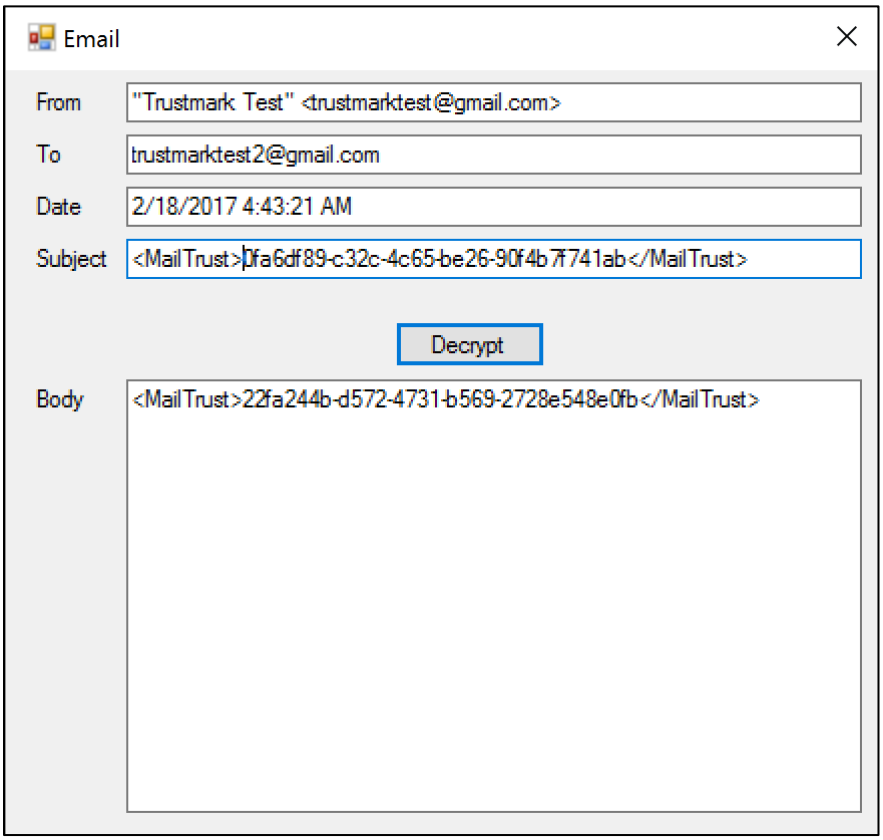


Figure 5.3 – MailTrust Example Client Before Decryption

Figure 5.3 diagrams the email-viewing pane of the example MailTrust client that was created to test the MailTrust architecture. The client connected to the Gmail servers over standard SMPT and IMAP protocols over SSL by utilizing a .NET email connection DLL provided by Chilkat Software [50]. This component ensured that the prototype client conformed to SMTP/IMAP standards. Full screenshots of the email sending, viewing, and decrypting process in the example client can be viewed in APPENDIX B.

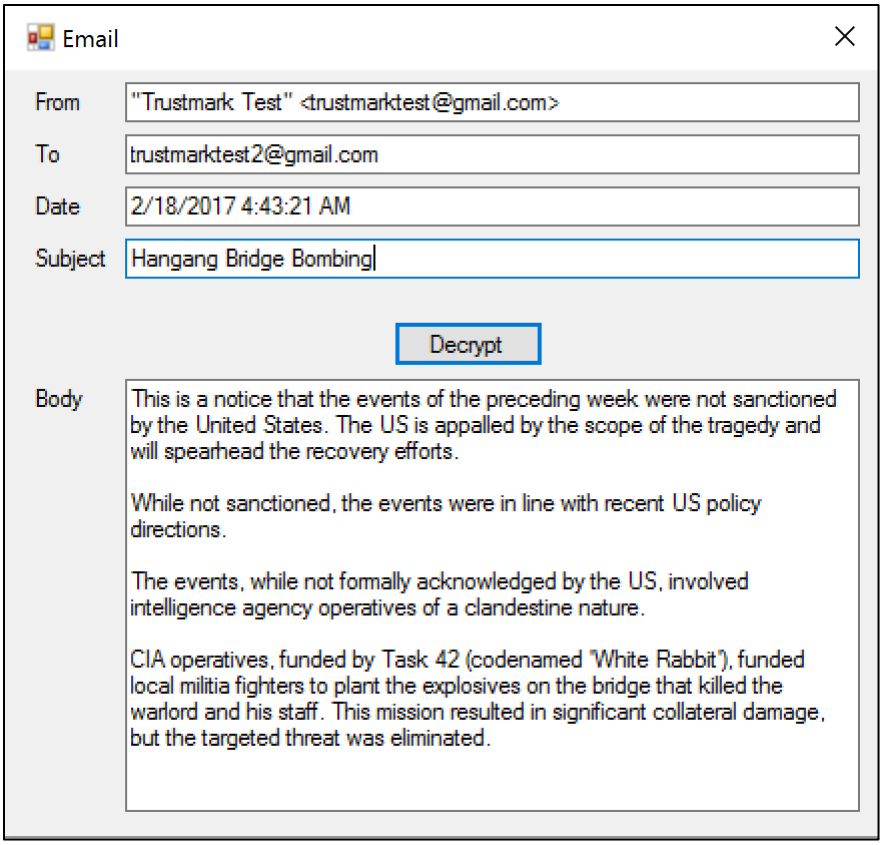


Figure 5.4 – MailTrust Example Client After Decryption

The prototype MailTrust email client utilizes a manual decryption/message retrieval button, shown in *Figure 5.4*, to provide a mechanism for easy testing of performance. When the user clicks the Decrypt button, the user's information is sent to the sender's MailTrust server for verification along with the handles to the MailTrust message content. If the user's information validates successfully, then the content is securely sent back to the client. A detailed examination of this message retrieval process is documented in the next section of this chapter.

In a real-world production system, the message retrieval functionality would be seamlessly integrated into the client. Plugins could be created for existing email clients that support a plugin architecture. According to Litmus Labs, as of January 2017, mobile devices such as Android and Apple iOS represented 56% of email usage, desktop at 17%, and webmail at 27% [51].

These data were compiled by Litmus tracking the opening of 1.25 billion emails across many different platforms. While desktop usage is only ~17%, it represents the majority of existing secure email transmissions due to the extensible support mechanisms available on a desktop platform. Implementing MailTrust in a production environment could be done in Microsoft Outlook, which represents 48% of all desktop email client usage and a vast majority of secure email usage, supports open plugin development through the standardized VSTO (Visual Studio Tools for Office) interfaces [52]. Outlook plugins using VSTO would enable MailTrust to work in an existing environment by simply adding a plugin to Outlook [53].

Mobile device email applications for Android and iOS could be created using standard mobile device languages or through multi-device development platforms, such as Xamarin that supports multiple target mobile platforms from a central development codebase [54]. Having clients readily available for multiple platforms is a key step to increase the adoption rate of MailTrust. Ultimately, it would be ideal if the native mail clients supported MailTrust as a standard messaging protocol.

The prototype MailTrust client shows that the feasibility of creating a real-world client that is able to be integrated into an already existing email ecosystem. Future work in optimization of the user interface is needed to provide a tool that users would accept. Ideally, the process for sending emails within MailTrust would be completely transparent to the user, with the exception of the steps required to choose the protection that needs to be applied to the content. Further research into intelligent security-level suggestive user interfaces is ideally needed to increase user acceptance and adoption of MailTrust.

Detailed Message Flow Process

Messages within the MailTrust architecture undergo multiple steps of security measures to ensure that the intended security controls are maintained throughout the message lifecycle.

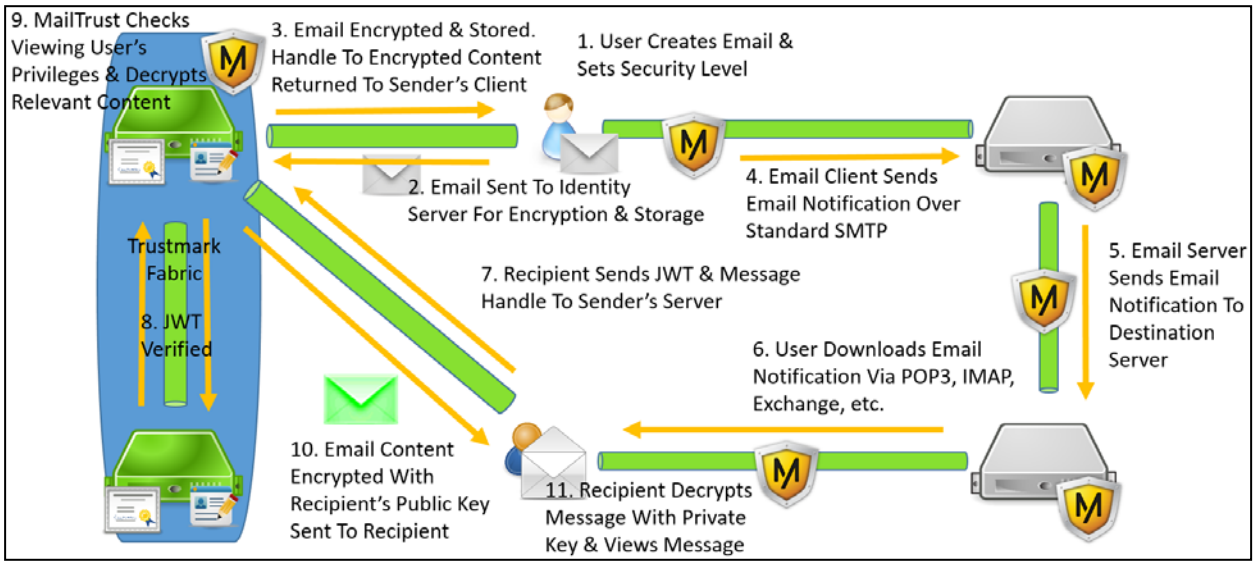


Figure 5.5 - MailTrust Message Flow

Verification procedures at each key data access point ensure that only the intended authorized recipients are allowed to retrieve the secured information. In order to fully describe the message flow process within MailTrust, this section breaks out each step and describes the processing and methodology contained within.

Figure 5.5 above shows each logical step within the MailTrust architecture. Each step is described in detail in the following sections.

1 – Message Creation

The initial creation of a MailTrust message follows the same methodology that takes place in any email content creation. Since MailTrust uses SMTP as its notification transmission framework, an email body and the recipient's email address are required. In addition to these base items, MailTrust requires access attributes be associated with each message. The attributes

can be bundled packages of attributes such as “confidential,” that may correspond to multiple underlying security requirements or a message can require only that a specific user’s signature be required to guarantee delivery. The MailTrust architecture allows for any Boolean combination of security and access controls but does not dictate any specific set of combinations of access control attributes. Instead, the security levels are dictated based on business requirements on a per-MailTrust basis. The control requirements for the DoD will be significantly different than that of a business or individual, so MailTrust is agnostic to the control requirements but instead enables a dynamic method of any combination of security requirements through the use of Trustmarks.

After a message content has been written and the targeted user(s) have been entered, the security level(s) of the content are then applied to the message through the client user interface. Future research into the optimal way to present a user interface for the assignment of security control levels is needed as that is beyond the scope of this current research. For the prototype system, security levels for the messages were hard-coded into the user interface. Each user of the system, detailed in APPENDIX C.1, had pre-assigned role attributes. For example, the user “Trustmark Test2” had roles of *Unclassified*, *Classified*, and *Secret*, but not *Top Secret*, so that user will be denied access to any *Top Secret* content.

2 – Transmission of Message to MailTrust

After the message has been authored and tagged with the appropriate security attributes, the message is then securely transmitted to the sender’s MailTrust server. This transmission is done via a Web Service that is exposed by the MailTrust server. The “StoreMessage” web method takes as input the message content, the desired access control requirements, as well as a JSON Web Token (JWT) that represents the identity of the sending user. The MailTrust server

verifies the identity of the sender's JWT by checking the JWT message signature against the identity provider for that MailTrust system. In the prototype system, the IDP and MailTrust servers were bundled together for ease on implementation, but in a production system they would likely be separate systems for scalability, system integrity, and integration with legacy platforms.

3 – Message Encryption and Storage

Once the sender is verified via their JWT signature, the message content is then encrypted and stored within the MailTrust Server. Encryption of the message takes place by using the identity signature of the sending user as the key for the encryptor function. Since only the sender and the sender's identity provider have access to the signature of a sender, the content within MailTrust is uniquely encrypted and safely stored on a per-sender basis. The MailTrust server encrypts the content of the message and stores the encrypted content, the username of the sender, the message dissemination security logic, and a globally unique identifier (GUID) that is generated by the MailTrust server that acts as a retrieval handle to the message.

Upon the successful encryption and storage of the message, the GUID handle to the message is returned to the client. A GUID is a 128-bit number that is represented by 32 lowercase hexadecimal (base 16) digits. GUIDS are commonly in the form of xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx where N indicates the UUID variant and the four bits of digit M indicate the UUID version. The hyphens are not part of a core GUID but increase readability and notation of the M and N version identifiers. With the hyphens, GUIDs can take up to 36 character space of storage. A version 4 pseudo-random generated GUID has approximately 5.3×10^{36} possible values and 103 trillion GUIDs would have to be generated for there to be a 1 in a 1 billion chance at a duplicate [55]. Due to the high probability of a lack of collisions, compact size, and

coupled with the ease and standardization of generation, GUIDs are the perfect choice for MailTrust message handles.

It is important to note that the recipient of a message is not uniquely stored for a given message. This allows a message to be forwarded to other recipients who have the valid authorization to view a given message. Since the dynamic security profiles for a message can be based on any Boolean logic combination of attributes, users can enforce polices that dictate that only a specific user can view a message. This would prevent even authorized individuals from viewing a message if it were forwarded to them if the sender enforces a single-recipient security profile.

4 – Transmission of Message Handle

After the message is encrypted and stored, the handle to the message is returned to the sender's MailTrust client. The client replaces all message content with the message handle in an XML tagged form so that other MailTrust clients can appropriately identify a MailTrust message from other email content. The MailTrust handle takes the form of the following tag:

<MailTrust>GUID</MailTrust>

Where GUID changes each time with the handle that is returned from the MailTrust server. Once the content has been replaced with the XML handle tags, the MailTrust server then sends the message over standard SMTP. In the prototype client, SSL encrypted SMTP was used but is not required. It is recommended to use SSL protected SMTP to ensure that the transmission of the content cannot be captured through network sniffing tools. While the message handle of a MailTrust transmission is useless itself, the existence of secured communication from a machine on a network could raise the attention of an attacker.

5 – Destination Server Routing

Once the message-handle-only MailTrust message is sent over SMTP, the sender's chosen SMTP server, Gmail in the prototype system, routes the message content to the destination recipient's email server. The server uses the mail exchange (MX) address of the recipient's email domain that is retrieved through domain name service (DNS) to determine the appropriate email server to route. An email address in canonical form is User@Domain.TopLevelDomain. DNS first examines the TopLevelDomain (TLD) suffix and contacts the domain registrar(s) associated with that TLD as determined by the Internet Corporation for Assigned Names and Numbers (ICANN). Once the official registrars associate with the TLD are known, the DNS registrar for the Domain is then looked up. The MX record for the Domain is then used to route the message to the destination recipient's mail server. DNS records are often cached at many levels, so the lookup times for all of these steps are very fast.

6 – Message Retrieval

After the message handle has been sent to the recipient's mail server, Gmail in the case of the prototype system, the message handle is then available for email client download. The prototype MailTrust system used Internet Message Access Protocol (IMAP) over SSL to download the message handle. While IMAP was used in the prototype system, any email retrieval protocol (POP3, Exchange, etc.) is allowable within the MailTrust architecture. Irrespective of the retrieval protocol used, it is highly recommended that SSL encryption be applied to the retrieval transmission. For the same reasons outlined in the transmission section, tipping off attackers of MailTrust protected content use could point attackers to machines/users. After the message has been downloaded from the recipient's server, the recipient's MailTrust client then identifies a MailTrust-enabled message by the XML tags within the body of the email. In the case where the

recipient's client is not MailTrust-aware, the content would just appear in the message body as normal text and would not be usable by the recipient. Since no sensitive content is transmitted over SMTP, MailTrust ensures central control over the protected content.

7 – Message Requesting

When a MailTrust-enabled client receives a MailTrust message notification through SMTP, the client examines the sender's email address for appropriate domain routing information. Using a DNS lookup similar to that of SMTP for message sending, the user's client looks up the MailTrust (MT) DNS records for the sender's domain. This MT record points to the sender's MailTrust server and provides a pathway for the recipient's client to request the message content. Once the client has the address of the sender's server through the MT record, the recipient's client then calls the GetMessage web method of the sender's MailTrust server. The client initiates this call by passing the web method two parameters, the GUID message handle and a JWT that represents the recipient's identity.

The user's JWT is signed with the user's private key as well as a signature by the user's identity provider. These cryptographic signatures provide the verification items needed for the sender's server to appropriately verify the identity of the claimed recipient.

8 – Trustmark Verification

Upon receipt of a GetMessage request from a user, a MailTrust server examines the JWT provided by the requesting user. The JWT, at a minimum, contains cryptographic signatures representing the user's identity as well as the identity of the user's identity provider. If the signature of the identity provider is not one that is trusted by the server, then the MailTrust server rejects the request.

The identity is verifiable since the MailTrust server contains the cryptographic Trustmarks issued to it by authorities that it trusts within its overall trust fabric. A trust fabric is a set of agreed upon Trustmarks between organizations that wish to communicate securely. Trust fabrics can be either explicit or implicit in nature. In an explicit model, Organization X would specify that it trusts the security principles of Organization Y and vice versa via a priori certificate key exchange. The implicit model is more scalable and is implemented by Organization A trusting a set of pre-defined Trustmarks, say {X,Y,Z}. If Organization B needs at a minimum Trustmarks {X,Y} to trust the security practices of an organization, then Organization A would be allowed to communicate with Organization B since A meets B's minimum security requirement. A set notation example of this trust fabric model can be defined as:

- For a given set of Trustmark requirements TR_B, if TR_B is a representative subset of available Trustmarks TA_A ($TR_B \subseteq TA_A$) then $\mathbf{\forall x\{x \in TR_B \rightarrow x \in TA_A\}}$.
- For a given set of Trustmark requirements TR_A, if TR_A is a representative subset of available Trustmarks TA_B ($TR_A \subseteq TA_B$) then $\mathbf{\forall x\{x \in TR_A \rightarrow x \in TA_B\}}$.
- If $TR_A = TR_B$ then the two sets of Trustmark requirements are considered equivalent and two organizations can commence with bidirectional communications.

Bidirectional trust fabric communication is not a requirement of MailTrust, but is needed if more than one-way communication is desired. The cryptographic trust fabric model based on Trustmarks is the key to MailTrust's ability to scale without the need of one-to-one or many-to-many key exchange models that do not scale well.

If the user and the identity provider do not have verifiable electronic signatures that correspond to the security Trustmark requirements established by the MailTrust server then the request is also rejected. If the signatures are verifiable, then an out-of-band request is made to the

recipient's identity provider for final verification. The identity of the recipient is queried by the MailTrust server by providing the JWT to the recipient's MailTrust server from the sender's MailTrust server. During this request, the sender's MailTrust server asks the recipient's server to provide applicable user attributes associated with the recipient. The request also asks for the public key that corresponds to the recipient user. After the identity is verified through the trust fabric and the public key is retrieved, the message can then be evaluated and processed.

9 – Message Authorization

One of the key benefits of the MailTrust architecture is the ability to dynamically process user attribute assertions to enforce content security access controls. Once the sender's server receives the attributes that correspond to a requesting recipient, it then processes those attributes. Each attribute is signed by Trustmarks issued to the recipient's server by an authorized Trustmark provider. If the sender's server and the recipient's server have a trust fabric that include the same Trustmark provider's trust fabric, then they can trust that authenticity of the attribute assertions made by each identity provider. This operates in a similar manner to the way root certificate stores operate in PKI. If a user's machine trusts certificates issued by a specified provider, indicated by the existence of a provider's public certificate in the root certificate store of a computer, then interactions with websites and other applications protected by certificates issued by that provider will be automatically trusted by that machine.

Once a level of trust has been determined based on the validity of the signatures of the user's Trustmark signed attributes, those attribute assertions can then be trusted by the sender's server. The attributes are then applied against the Boolean logic that corresponds to the security requirements of the message stored on the MailTrust server. If the attributes required by the

logic are met by the recipient, then the message can be returned to the recipient. Before returning the message to the recipient, the content must be protected to ensure secured delivery.

10 – Target Recipient Encryption

In order to ensure that only the intended recipient can view the message content, the sender's MailTrust server takes the message content and encrypts the content using the recipient's public key that was provided out-of-band by the recipient's MailTrust/Identity server. This encryption by the user's public key ensures due to the properties of asymmetric key pairs that only the user with the corresponding private key can view the encrypted content. This step functions in the same manner as the underutilized S/MIME encryption method.

```
1   using (var cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write))
2   {
3   cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);
4   cryptoStream.FlushFinalBlock();
5   // Create the final bytes as a concatenation of the random salt bytes, the random iv bytes and the cipher
6   bytes.
7   var cipherTextBytes = saltStringBytes;
8   cipherTextBytes = cipherTextBytes.Concat(ivStringBytes).ToArray();
9   cipherTextBytes = cipherTextBytes.Concat(memoryStream.ToArray()).ToArray();
10  memoryStream.Close();
11  cryptoStream.Close();
12  return Convert.ToBase64String(cipherTextBytes);
13  }
```

Figure 5.6 – Encryptor Code [56]

11 – Final Message Decryption and Viewing

In the last step of the MailTrust message process, the target recipient receives the message from the sender's MailTrust server that is encrypted with the recipient's public key. Using their private key, the recipient is able to decrypt the message.

```
1    using (var cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read))
2    {
3    var plainTextBytes = new byte[cipherTextBytes.Length];
4    var decryptedByteCount = cryptoStream.Read(plainTextBytes, 0, plainTextBytes.Length);
5    memoryStream.Close();
6    cryptoStream.Close();
7            return Encoding.UTF8.GetString(plainTextBytes, 0, decryptedByteCount);
8    }
```

Figure 5.7 – Decryptor Code [56]

As seen in *Figure 5.7*, on line 1, a .NET CryptoStream is created using a memoryStream of text corresponding to the encrypted message content. A decryptor variable is supplied that includes the private key of recipient user. The .NET crypto libraries then apply RijndaelManaged AES 256bit decryption to decode the message content. Since only the intended recipient has the corresponding private key, it is guaranteed that only the correct user can appropriately decrypt the message content. After the message is decrypted, the content is then viewable within the MailTrust client. The client does not allow the content to be saved locally, so the server maintains full control of the message content. Future implementations of a MailTrust client could implement countermeasures to further control message content, such as disabling copy/paste and disabling the ability for a system to screen capture content. If these measures are implemented, then one of the only remaining attack vectors after message decryption resides in the ability for someone to take a picture of the content on the screen from another camera-equipped device. Research into ways to prevent those attack vectors could include utilization of a device camera to detect another camera and blank out the message accordingly. While no system is 100% safe from any attack vector, MailTrust provides an assurance architecture that protects the content from the sender to the recipient and maintains message security throughout that lifecycle.

The security and control measures that MailTrust offers provides unique and compelling solutions to the gaps that exist with secure email systems. The ability to track and revoke messages,

having complete access to document control measures, and the client/server level security measures, all help ensure that the MailTrust architecture meets the most stringent requirements of secure government agencies and industry. The unified control measures and features that MailTrust puts in place provides a number of advantages above existing solutions. These measures have a small impact in performance, which is discussed in detail in the next chapter; but the aggregate overhead is not noticeable to a user as increased transmission times in terms of seconds is a normal expectation of email communications users.

CHAPTER 6 - SIMULATION RESULTS

In this section we examined the performance overhead introduced by the MailTrust system. Performance tests on the MailTrust architecture were designed to simulate the differential between the MailTrust implementation and that of a traditional email system. We evaluated the message size differential between encrypted and non-encrypted messages and we also evaluated the encryption time to secure the messages. Simulations were performed using the prototype MailTrust system detailed in Chapter 5. Evaluations of encryption overhead, message storage space, and message transmission size were all areas of focus for the simulations.

Encryption is a key component of MailTrust, and it is maintained for data at rest and in transit. Data in transit is protected through a combination of traditional SSL and PKI signing of the message requests and assertions. Data that are persisted to disk on the MailTrust servers are encrypted with 256bit AES encryption utilizing a salted hash that is iterated over 100,000 times to mitigate the risk of brute force attacks. To test the overhead produced by the encryption process, a random content generator was constructed to create worst-case scenarios, whereby compression optimization techniques would not be a factor and the encryption algorithms would have to function at full capacity. To generate the message bodies for testing, the following character set was used for random selection:

```
abcefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ,-
.!'"()%$?@;',&#
```

Figure 6.1 – Random Email Generation Set

The message generation algorithm generated random messages that increased in 50 characters in size per message. The messages were UTF-8 in format to comply with common email formats. UTF-8 character storage can vary between 1 and 4 bytes per character, depending on characters used. Since the sampled character set only resides within the ASCII character set domain, that byte representation only requires one byte of storage in accordance with Table 5.1 below.

Table 6.1 - UTF-8 Character Storage Description

Binary	Hex	Storage Bytes	Comment/Purpose
0xxxxxxx	0x00..0x7F	1-Byte	ASCII Character Set
10xxxxxx	0x80..0xBF	1-3 Bytes	Continuation bytes, if using other characters below
110xxxxx	0xC0..0xDF	2-Bytes	All European plus some Middle Eastern
1110xxxx	0xE0..0xEF	3-Bytes	Multilingual plane & private-use
11110xxx	0xF0..0xF4	4-Bytes	Math symbols, emoji & other old scripts

Each progressively larger message was encrypted via the process 256bit AES encryption described in Chapter 4 and the resulting message sizes, as well as time taken for the encryption process, were logged. As seen in *Figure 6.2*, an average 34% overhead (plus a base differential of 109 bytes) exists between unencrypted versus encrypted message sizes. As message sizes increase the gap widens but in a linear progressive fashion. For a given message byte count B_{count}, the delta byte differential Δ_{comp} can be calculated for future message compression sizes using the following function:

$$\Delta_{comp} = (B_{count} * 0.3372) + 108.81$$

Figure 6.2 – MailTrust Storage Overhead Equation

According to a 2009 study by ActivityOwner.com [57], the average email length is 3150 characters. Applying this metric to the above function results in an estimated byte differential of 1170.99 bytes. This represents approximately 37% overhead required for properly encrypted storage of an average length email message.

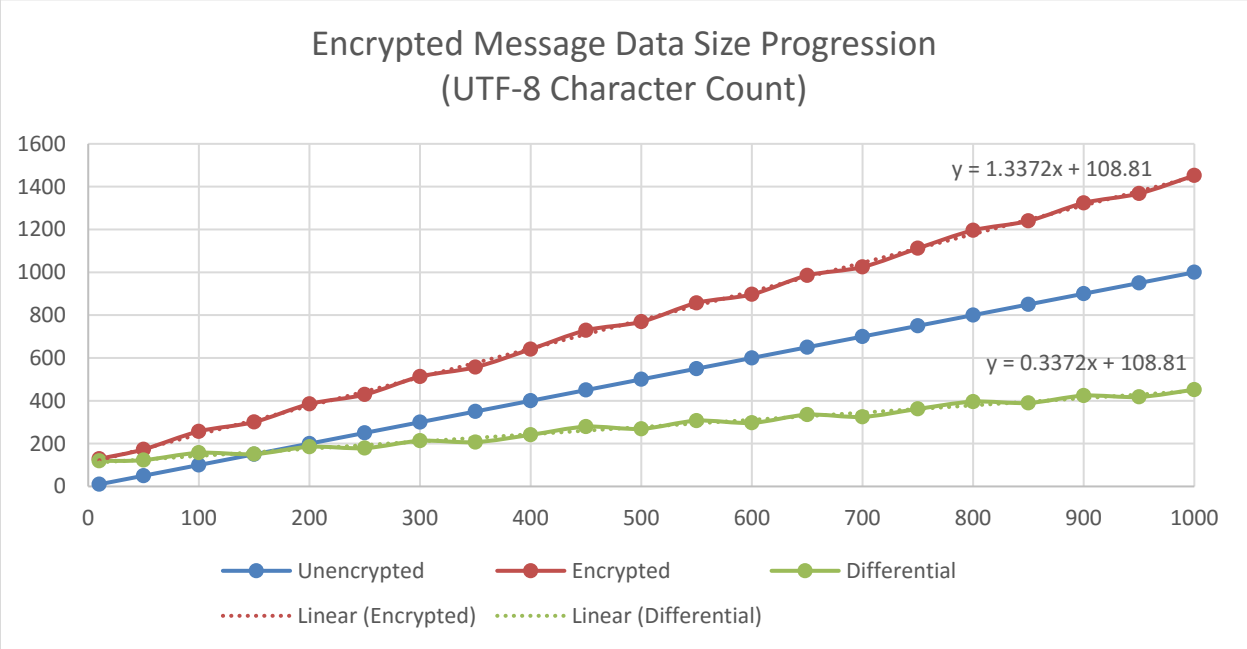


Figure 6.3 - Encrypted vs Unencrypted Message Size

While a 37% storage overhead exists for message storage, since none of the MailTrust messages are ever stored client side, the effective storage impact both to the server and system-wide are actually a net savings. A single message to three recipients in a traditional email system has four copies of the message (one in the sender's sent folder and one in each of the recipients' inbox). If the message were the average size of 3150 bytes then that email communication would have required 12,600 bytes of storage just for the content. In the MailTust architecture, that same message content would occupy only 4,321 bytes, which results in a 291% increase in

storage efficiency. *Figure 6.3* diagrams the message size overhead of encrypted verses non-encrypted message sizes.

The strength of the AES encryption used is directly tied to the number of times that the salt hash is reiterated. An analysis of iteration overhead was performed on the MailTrust architecture to evaluate the overhead introduced by varying message sizes and salt iterations. Messages of sizes 100, 250, 500, 750 and 1,000 characters were benchmarked against salt iterations of 1,000, 10,000 and 100,000 rounds. Password-Based Key Derivation Function 2 (PBKDF2), which is a pseudorandom function for generation of a derived key passed to the AES encryption function, takes the random salt and hashes over it a set number of passes to increase randomness and reduce the chances of future brute force attacks against the protected content. When PBKDF2 was introduced in calendar year 2000, 1000 iterations were the recommended count. As CPU power has increased, so have the recommended iteration counts. In 2005 Kerberos recommended 4096, and LastPass in 2015 [58] increased their iterations to 100,000 for server-side storage.

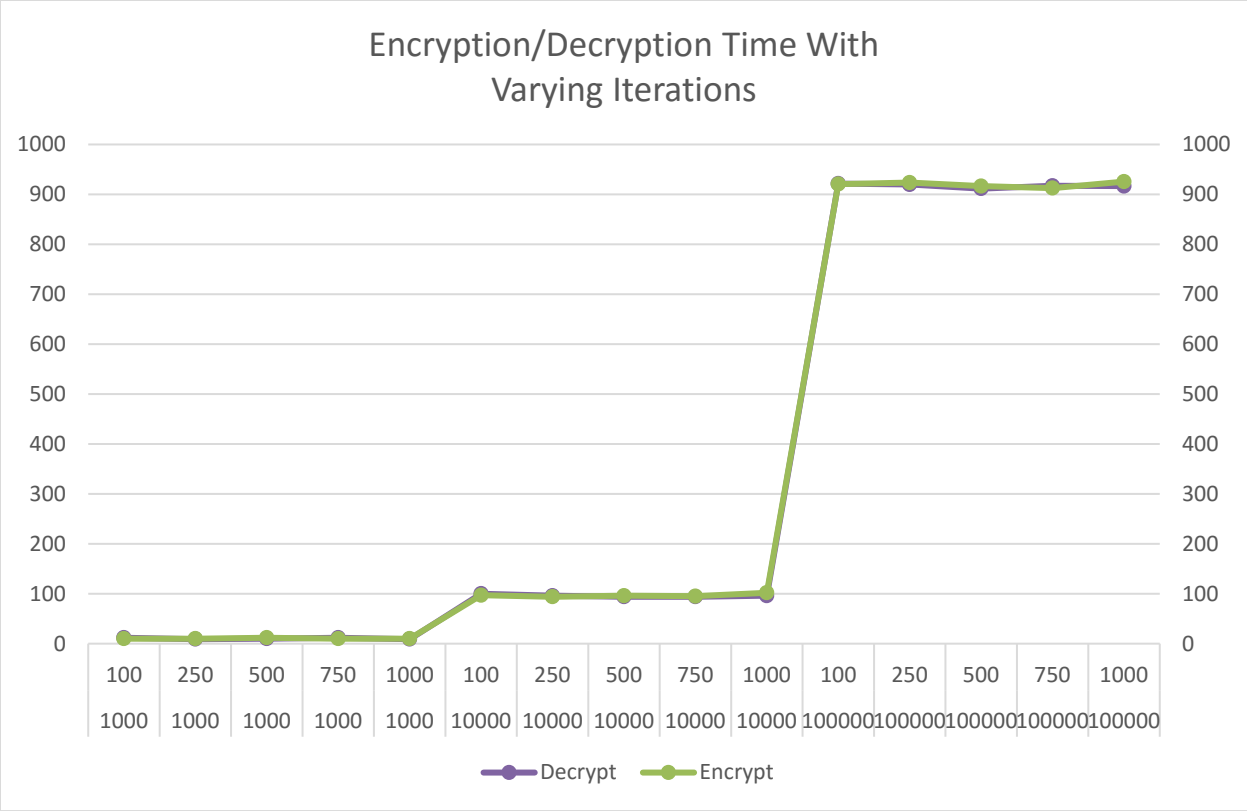


Figure 6.4 – Storage Message Size (Bytes) vs. Encryption Time (in milliseconds)

An analysis was performed against varying length message sizes in the context of multiple iteration lengths. As shown in *Figure 6.4*, 1000 hash iterations were performed on average in under 10ms while 10,000 iterations required approximately a 10x increase in time at approximately 100ms. This linear progression increased at 100,000 iterations, whereby the computation time took slightly less than 1 second at 900ms. The overhead for both decryption and encryption are the same for MailTrust, so the impact is potentially applicable on both the message sending and message retrieval sides. However, the message sending process happens in an asynchronous manner. When an email client sends a message to a MailTrust server for storage, the MailTrust server generates the GUID handle to the message immediately and then returns the handle to the client. In a separate thread, the MailTrust server then takes the GUID, message content, security parameters, and sending user ID and persists them to the MailTrust storage fabric.

The overhead required for the PBKDF2 hash iterations was found to be linear with the number of iterations, irrespective of message length. Both encryption and decryption times were consistent irrespective to the number of times the salt hash was reiterated. *Figure 6.4* contains both encryption and decryption times; but due to the tight correlation between the two values, they are virtually indistinguishable.

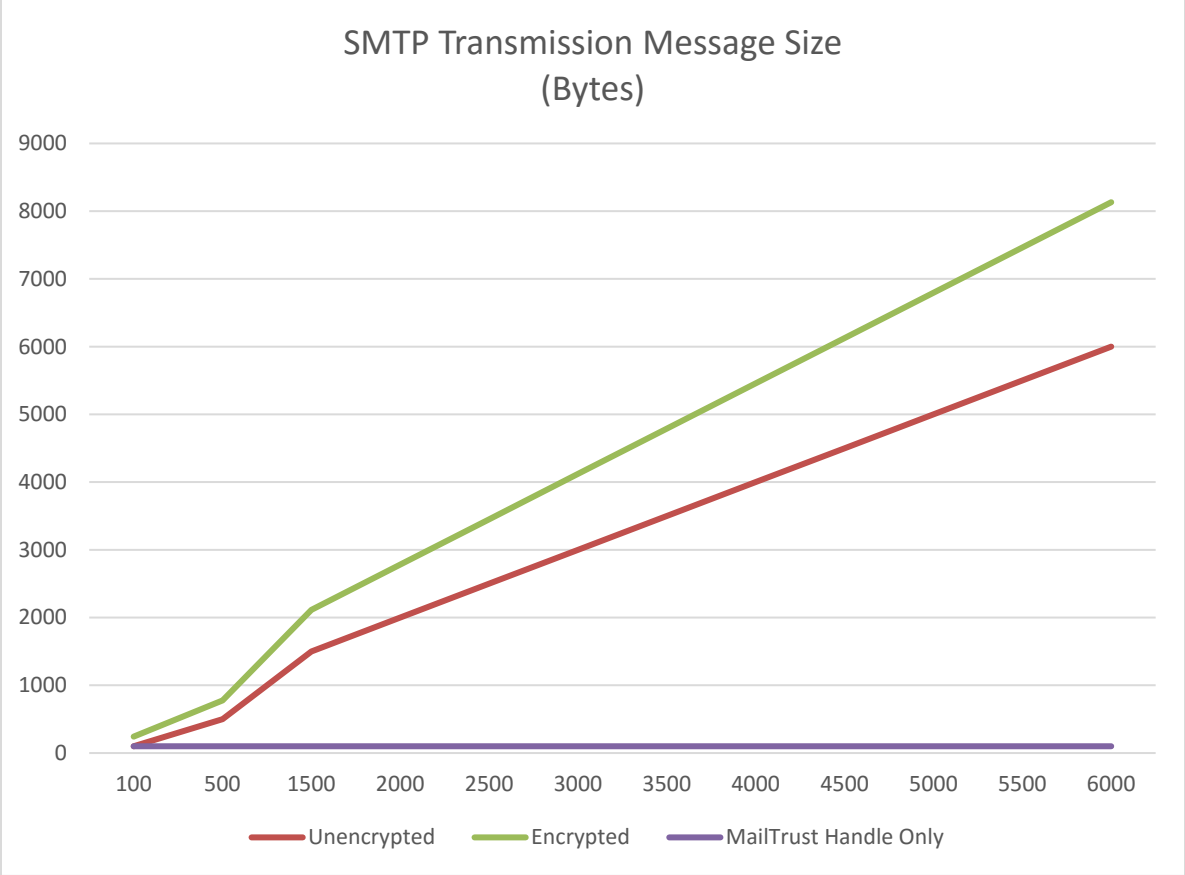


Figure 6.5 – SMTP Transmission Message Size (Bytes)

In a traditional email message system, messages are transmitted to all recipients in either an encrypted or unencrypted form. In a MailTrust system, only the message identifier is transmitted over SMTP and the secure content is transmitted directly from the source server repository to the consuming client after proper client authentication. The transmittal of only the mes-

sage identifier reduces the message notification time and storage requirements across every device and server within the ecosystem. As seen in *Figure 6.5*, the SMTP message transmission size remains consistent with MailTrust, irrespective of size of the encrypted content that is stored on the server.

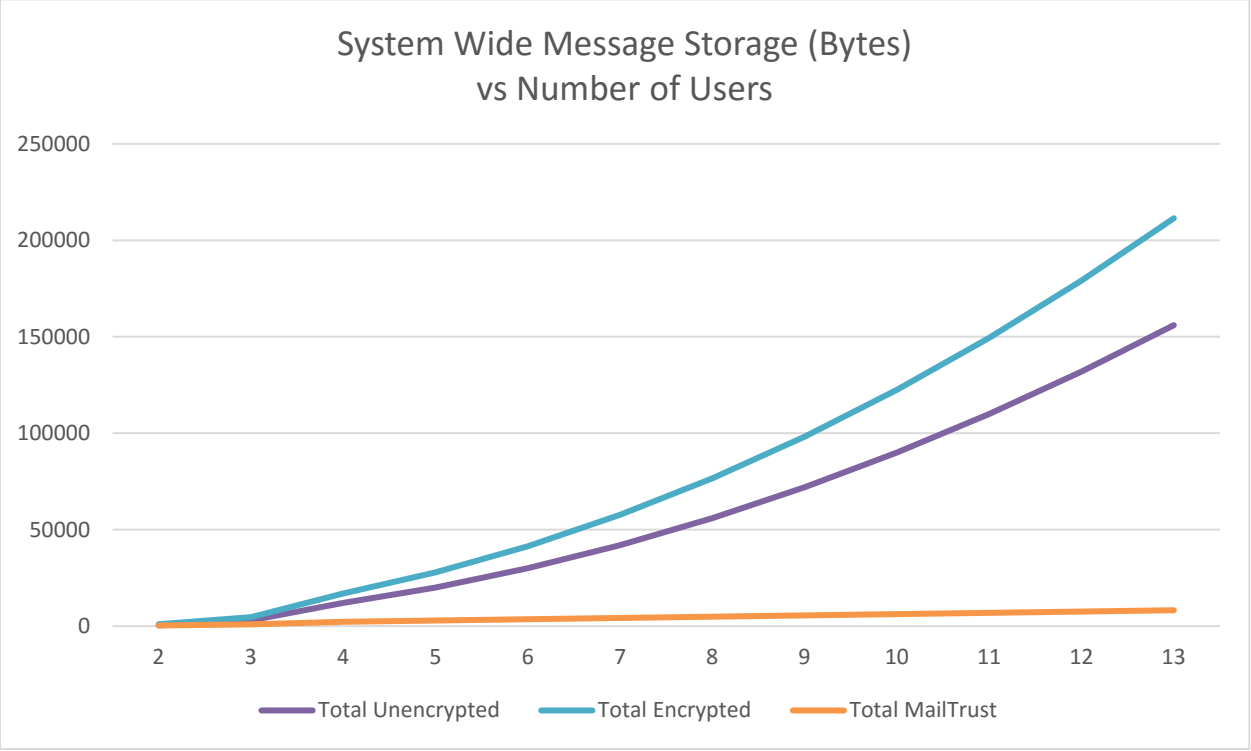


Figure 6.6 – System Wide Message Storage

Since a MailTrust system only transmits the message handle, the overall system storage of messages is drastically reduced. In a system where each user has their own device and email server, the MailTrust message storage requirements remains linear, and it is only tied to the size of the encrypted message that is stored on the sender's MailTrust server. *Figure 6.6* shows a graph of the increasing storage requirements based on increasing number of users within the system (email recipients). The MailTrust architecture results in a linear increase in system storage

requirements that are tied directly to the sending message size. Traditional PKI encrypted email (or unencrypted email), however, results in a polynomial increase in storage requirements.

While MailTrust introduces system overhead from its enhanced encryption and verification measures, the overhead is not readily visible to a user. Since the approximate 1-second time to encrypt the content happens asynchronously, the overhead for sending a message is not visible to a user. Additionally, since the message handler only content is smaller in size compared to the full message, transmission times will inherently be reduced. The time required to download the message handle on the recipient's side will also be reduced, but overhead is introduced during the message retrieval phase from the MailTrust server. Intelligent coding of a MailTrust email client can eliminate any perceived overhead by this retrieval process as the client can keep in memory only the downloaded message content. The only time where a user would experience noticeable overhead would be the retrieval of old MailTrust content, which would incur the 1-second decryption time in addition to the user verification and content download. Overall, in an appropriately constructed email system, a user would experience no more than a few seconds of delay while retrieving content and that is on par with existing email systems, especially for older archived content.

Since the overhead introduced by MailTrust does not have a significant impact on user experience, the overall system performance of MailTrust is within expectations. The system-wide reduction in content storage has a significant benefit to all and the retention of control of the message lifecycle is a significant step forward in email security.

CHAPTER 7 – CONCLUSIONS

Currently adopted email security standards date back more than 30 years, and while a number of enhancements have been proposed, few have gained traction. Our society has become reliant upon email as a *de facto* standard communication medium. Email has in many cases supplanted the role of fax machines for the secure digital transmissions of documents from point to point. However, the security control mechanisms that safeguard email are severely lacking when it comes to information assurance and the controlling of the dissemination of information. These large gaps create a clear need for a new security-focused email methodology whereby the authenticity and security of messages.

PKI-based email security, which is the most commonly used secure email technology, is limited in scope by design, and it only addresses the desirable properties of confidentiality and integrity. However, PKI-based solutions alone do not address issues of end-point authentication that are critical to the transmission of information, such as military intelligence information that is sensitive or "classified." The solutions proposed in existing literature provide a starting point to address all three security properties of confidentiality, integrity and end-point authentication, but all suffer from scalability problems when trying to adapt those systems to operational reality.

The MailTrust architecture described in this work provides many advancements over standard email or the commonly used PKI based S/MIME email protection. Specifically, a new encryption process is outlined in Chapter 3 for emails based on AES encryption coupled with the

sender's user identity hash from an identity server. A new and innovative use model for Trustmarks is also outlined in Chapter 3 that defines an easily scalable architecture for the deployment of a MailTrust system across multiple organizations. MailTrust also implements Trusted Identity transformation of FIPS 199, DoD 5200.01 and NIST 800-53 to their Trustmark-equivalent user assertion paradigms for the implementation of Email non-repudiation controls. Chapter 3 also outlines dynamic email identity management via Trustmarks in a multi-federation environment and a secure email content information assurance process. Identity coordination with external organizations is also implemented through the MailTrust architecture outlined in those sections.

MailTrust implements a number of security and control methods that are outlined in Chapter 4. Message revocation through the use of single-source secure message repositories that facilitate editing and message revocation are presented, and a XACML-based email document security model that processes Trustmark-signed user assertions for content control is also outlined. Chapter 4 also covers server security and control procedures that are available in the MailTrust architecture, including transmission control measures that ensure the proper delivery of the secured content to the intended email recipient.

An evaluation of the MailTrust performance overhead is examined in Chapter 6 to determine the system-wide impact of implementing MailTrust. Performance evaluations were accomplished on an implementation of the MailTrust architecture that was created as part of this effort.

Email, particularly on mobile phones, has become a primary *de facto* communication mechanism in our society [51]. The ability to purposefully or accidentally distribute sensitive information through email presents a large security hole for both industry and government. A new email security model is needed to support the growing needs and use case scenarios for secure email. Principles such as identity-based crypto, attribute-based authentication and content

control, and system policies, coupled with the best points of existing PKI were leveraged to construct MailTrust as the next-generation secure email architecture.

There are a number of gaps within the existing S/MIME PKI architecture and those gaps are increasingly contributing to data breaches that are having far-reaching political and societal impacts [5]. MailTrust not only addresses those needs, but presents a solution that goes well beyond the deficiencies that exist within S/MIME. Features like message revocation, extensive non-repudiation controls, and integrated message read receipts can all be implemented seamlessly within an email ecosystem that is based on the MailTrust architecture. Since MailTrust utilizes SMTP for message handle transmission, it integrates seamlessly within existing email ecosystems. Non-MailTrust enabled email clients can still receive secure MailTrust notifications and non-secure emails, but they will not be able to download the secured content since they cannot appropriately authenticate. The adoption rate of MailTrust will be enhanced by its ability to work within existing email environments, and this co-existence will help facilitate multi-platform capability and spur email security enhancements.

The pervasiveness of email in our society has created a communication alternative to fax machines, physical mail, and other mediums that has changed the way in which we communicate. Some facets of our society, such as the transmission of medical records, still rely on legacy fax machines due to the inherent insecurity of email transmission. A new security-focused email architecture that addresses existing gaps and provides a scalable future is needed to facilitate the continued growth and secure use of email. MailTrust is a major step forward in meeting those demands by providing a platform on which future research and enhancements can be made. In addition to providing a secure email architecture, MailTrust is extensible to other domains where secure messaging, non-repudiation, and trusted identity are needed.

Future Research

There are a number of ways that the MailTrust architecture could be further expanded and enhanced in the future. In the course of this effort, a number of future enhancements and research directions were identified. In particular, there is strong potential to expand MailTrust to include multitier messages that respond with different content based on the attributes of the requesting user. A message architecture to support an email equivalent of a multilevel security (MLS) database model in the context of email messages is a natural extension of this research. In a multilevel secure database management system (DBMS), each user gains access directly or indirectly to only those data for which they have proper clearance [59], [60]. An information hiding, replacement and abstraction paradigm could be developed to support policy-based dynamic email content control within the MailTrust framework.

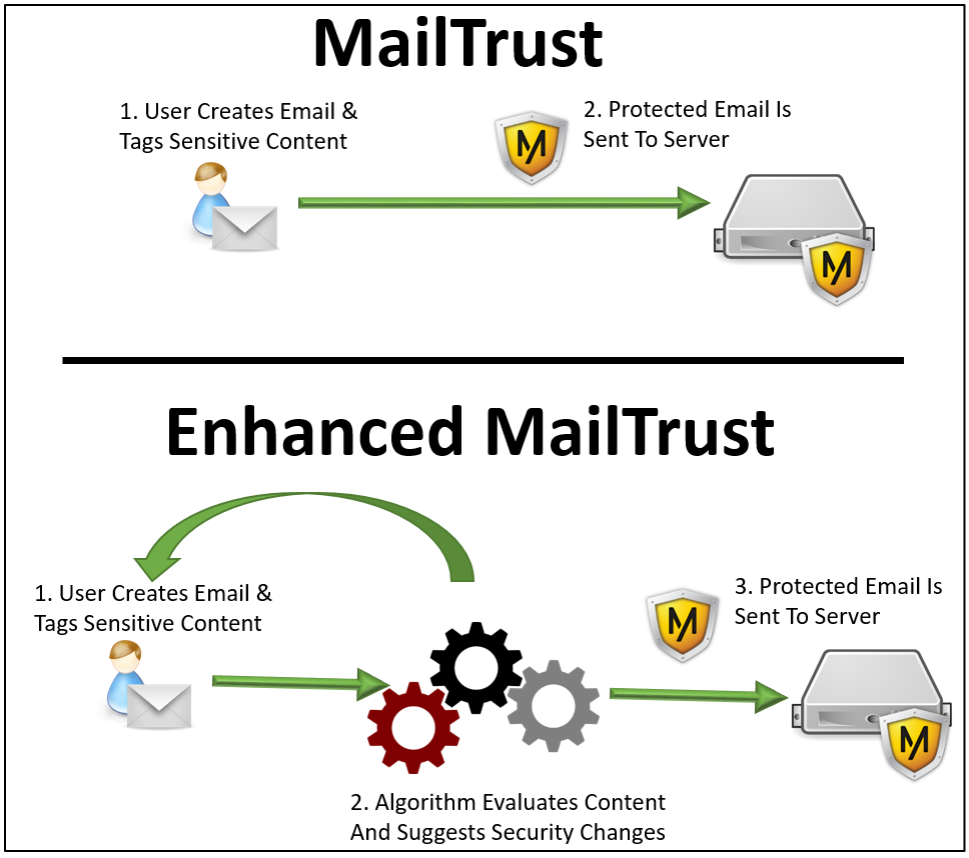


Figure 7.1 – Future Security Level Suggestion Process

There is also the potential for future research around a dynamic message attribute security suggestion algorithm. An algorithm, suggested by *Figure 7.1*, could be developed that analyzes document tagging from past emails and suggests appropriate document classification based on the current email content. There are a number of efficient document similarity and clustering algorithms in the literature that could be leveraged in this domain [61], [62]. This algorithm could enhance both the adoption rate of a MailTrust system as well as the overall security model by integrating a degree of artificial intelligence and further looping the user into the document security level rating process.

While this research primarily targeted the use of MailTrust in the context of government/DoD systems, the overall architecture is beneficial to industry and individuals as well who

wish to utilizes secured messaging with all of the inherent benefits that MailTrust provides. Furthermore, while MailTrust was designed to solve secure email solutions, the architecture could be easily extended to other domains. Systems that need secured message delivery in a mixed authentication environment, such as ecommerce, online file sharing, remote application data access, and network access controls to a remote network could all benefit from adaptations of the MailTrust architecture.

The MailTrust architecture solves many of the problems created by the gaps that exist with current email security solutions. MailTrust provides a more efficient storage solution, maintains centralized granular and revocable control over email content, and provides a pluggable architecture through which additional enhancements can be added in the future. As shown in this work, the MailTrust architecture delivers a next-generation email solution that is backwards compatible to, and can be fully integrated with, existing email and authentication ecosystems. These attributes position MailTrust to be a viable solution to solve the large gaps that exist in secure email for both industry and government.

The MailTrust research described herein has been vetted and approved by the Institutional Review Board (IRB) of the University of Alabama's Office of Research & Compliance. The corresponding IRB approval letter is included in APPENDIX F.1.

REFERENCES

[1] I. T. Union, “X.500 (11/1988),” vol. 8, 1988.

[2] P. Zimmermann, “Pretty Good Privacy--RSA public key cryptography for the masses.” June, 1991.

[3] J. Brooke, *“SUS-A quick and dirty usability scale.” Usability evaluation in industry*. CRC Press, 1996.

[4] J. Brooke, “SUS : A Retrospective,” *J. Usability Stud.*, vol. 8, no. 2, pp. 29–40, 2013.

[5] JOSH GERSTEIN and RACHAEL BADE, “22 Hillary Clinton emails declared ‘top secret’ by State Dept. - POLITICO,” 2016. [Online]. Available: http://www.politico.com/story/2016/01/22-hillary-clinton-emails-declared-top-secret-218420. [Accessed: 11-Feb-2017].

[6] CNN, “Dems open convention without Wasserman Schultz - CNNPolitics.com.” [Online]. Available: http://www.cnn.com/2016/07/22/politics/dnc-wikileaks-emails/. [Accessed: 11-Mar-2017].

[7] Office of the Director of National Intelligence, “Organization Structure of Office of National Intelligence.” [Online]. Available: https://www.dni.gov/index.php/about/organization. [Accessed: 11-Feb-2017].

[8] “Members of the Intelligence Community.” [Online]. Available: https://www.dni.gov/index.php/intelligence-community/members-of-the-ic#nsa. [Accessed: 11-Feb-2017].

[9] J. S. Hollywood and Z. Winkelman, “Improving Information-Sharing Across Law Enforcement: Why Can’t We Know?”

[10] P. P. Swire and C. Q. Butts, “The ID Divide: Addressing the Challenges of Identification and Authentication in American Society,” *Cent. Am. Prog.*, 2008.

[11] “Privacy Policy: Fair Information Practice Principles,” 2008.

[12] C. A. Lee, “Cloud Federation Management and Beyond: Requirements, Relevant Standards, and Gaps,” *IEEE Cloud Comput.*, 2016.

[13] “Control Use of Data to Protect Privacy Susan Landau,” *Sci.*, vol. 347, no. 6221, 2015.

[14] S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller, “How to make secure email easier to use,” *Proc. ACM CHI 2005 Conf. Hum. Factors Comput. Syst.*, vol. 1, pp. 701–710, 2005.

[15] “Exclusive: Big data breaches found at major email services - expert | Reuters,” 2016. [Online]. Available: http://www.reuters.com/article/us-cyber-passwords-idUSKCN0XV1I6. [Accessed: 11-Feb-2017].

[16] S. Ruoti *et al.*, “‘ We’re on the Same Page ’ : A Usability Study of Secure Email Using Pairs of Novice Users.”

[17] Usd, “DoD Manual 5200.01, Volume 2, February 24, 2012; Incorporating Change 2, March 19, 2013,” 2012.

[18] D. Ppoi and S. Directorate, “UNCLASSIFIED – CLASSIFICATION MARKINGS FOR TRAINING PURPOSES ONLY General Marking Requirements,” 2013.

[19] M. Blaze, J. Feigenbaum, and J. Lacy, “Decentralized Trust Management.”

[20] E. Gerck, “Secure Email Technologies X. 509/PKI, PGP, IBE and Zmail,” *Corp. Email Manag.*, vol. 12797, pp. 1–23, 2007.

[21] GlobalSign, “Secure Email - Digitally Sign & Encrypt Emails,” 2017. .

[22] IASE - Information Assurance Support Environment, “External and Federal PKI Interoperability,” 2016. [Online]. Available: http://iase.disa.mil/pki-pke/interoperability/Pages/index.aspx. [Accessed: 10-Feb-2017].

[23] P. Hoffman, “SMTP Service Extension for Secure SMTP over Transport Layer Security.”

[24] H. Orman, *Encrypted Email The History and Technology of Message Privacy*. 2015.

[25] A. Kapadia, “A case (study) for usability in secure email communication,” *IEEE Secur. Priv.*, vol. 5, no. 2, pp. 80–84, 2007.

[26] A. Kapadia, “A Case (Study) For Usability in Secure Email Communication,” *IEEE Secur. Priv.*, no. March, pp. 1–30, 2009.

[27] T. Chen and S. Ma, “A secure Email encryption proxy based on identity-based cryptography,” in *Proceedings - 2008 International Conference on MultiMedia and Information Technology, MMIT 2008*, 2008.

[28] J. H. Saltzer and M. D. Schroeder, “The Protection of Information in Computer Systems,” *Proc. IEEE*, 1975.

[29] A. Shamir, “Identity-Based Cryptosystems and Signature Schemes,” *Adv. Cryptol.*, vol.

196, pp. 47–53, 1985.

[30] D. Boneh and M. Franklin, “Identity-Based Encryption from the Weil Pairing,” *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, 2003.

[31] M. Baldwin, “Identity Based Encryption from the Tate Pairing to Secure Email Communications,” no. May, 2002.

[32] C. P. Masone, “Attribute-based, usefully secure email,” 2008.

[33] C. Masone and S. Smith, “Towards usefully secure email,” *IEEE Technol. Soc. Mag.*, vol. 26, no. 1, pp. 25–34, 2007.

[34] F. Paci *et al.*, “Minimal credential disclosure in trust negotiations.”

[35] A. Ghafoor, S. Muftic, and G. Schmölzer, “CryptoNET: Design and Implementation of the Secure Email System.”

[36] A. Mislove *et al.*, “POST : A Secure , Resilient , Cooperative Messaging System,” *EuroSys ’06 Proc. 1st ACM SIGOPS/EuroSys Conf. Comput. Syst.*, 2006.

[37] Z. Wu, J. Tian, S. Member, L. Li, C. Jiang, and X. Yangi, “A Secure Email System Based on Fingerprint Authentication Scheme,” *Intell. Secur. Informatics, 2007 IEEE*, pp. 250–253, 2007.

[38] A. Antón, D. M. Blough, E. A. Reddick, and P. Swire, “Trustmarks and Privacy,” 2014
.

[39] S. Quirolgico, V. Hu, and T. Karygiannis, *Access Control for SAR Systems*. 2011.

[40] GTRI, “The Trustmark Concept | GTRI NSTIC Trustmark Pilot.” [Online]. Available: https://trustmark.gtri.gatech.edu/concept/. [Accessed: 20-Feb-2017].

[41] “Scaling Interoperable Trust through a Trustmark Marketplace IDESG BOF Machine Readable Trustmarks,” 2014.

[42] Axiomatics, “pure-xacml,” *www.axiomatics.com*.

[43] D. W. Chadwick and G. Inman, “Attribute Aggregation in Federated Identity Management,” *IEEE Comput. Soc.*, no. May, pp. 33–40, 2009.

[44] B. Thuraisingham, “Information Sharing Strategies of the United States Federal Government and Its Allies and Our Contributions Towards Implementing these Strategies,” no. August, 2010.

[45] A. Cornellisen, “Covert Channel Data Leakage Protection,” Nijmejen, Netherlands, 2012.

[46] F. Information and P. Standards, “FIPS-197: Announcing the Advanced Encryption

Standard (AES)," … *Technol. Lab. Natl. Inst. Stand.* …, vol. 2009, pp. 8–12, 2001.

[47] Google, "SHAttered - SHA-1 Broken." [Online]. Available: http://shattered.io/. [Accessed: 24-Feb-2017].

[48] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[49] IdentityServer, "IdentityServer," 2017. [Online]. Available: https://identityserver.io/. [Accessed: 18-Feb-2017].

[50] Chilkat Software, ".NET Email Component by Chilkat." [Online]. Available: https://www.chilkatsoft.com/email-dotnet.asp. [Accessed: 19-Feb-2017].

[51] Litmus Labs, "Email Client Market Share and Popularity - January 2017." [Online]. Available: https://emailclientmarketshare.com/. [Accessed: 19-Feb-2017].

[52] Microsoft, "Office Solutions Development Overview (VSTO)." [Online]. Available: https://msdn.microsoft.com/en-us/library/hy7c6z9k.aspx. [Accessed: 19-Feb-2017].

[53] Microsoft, "Walkthrough: Creating Your First VSTO Add-In for Outlook." [Online]. Available: https://msdn.microsoft.com/en-us/library/cc668191.aspx. [Accessed: 19-Feb-2017].

[54] Xamarin, "Mobile App Development & App Creation Software - Xamarin." [Online]. Available: https://www.xamarin.com/. [Accessed: 20-Feb-2017].

[55] L. Zahn, *Network computing architecture*. Prentice Hall, 1990.

[56] Craig Phillips, "Encrypting & Decrypting a String in C# - Stack Overflow," 2015. [Online]. Available: http://stackoverflow.com/questions/10168240/encrypting-decrypting-a-string-in-c-sharp?answertab=oldest#tab-top. [Accessed: 15-Feb-2017].

[57] ActivityOwner.com, "How long is a typical email message? » ActivityOwner.Com – Getting Things Done with MindManager." [Online]. Available: http://www.activityowner.com/2009/03/14/how-many-characters-are-there-in-a-typical-email-message/.

[58] J. Siegrist, "LastPass Security Notification," 2011. [Online]. Available: https://blog.lastpass.com/2011/05/lastpass-security-notification.html/.

[59] S. Jajodia and R. Sandhu, "Toward a multilevel secure relational data model," *ACM SIGMOD Rec.*, vol. 20, no. 2, pp. 50–59, 1991.

[60] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. Shockley, "Multilevel Relational Data Model.," *Proc. Symp. Secur. Priv.*, pp. 220–234, 1987.

[61] H. Chim and X. Deng, "Efficient phrase-based document similarity for clustering," *IEEE*

Trans. Knowl. Data Eng., vol. 20, no. 9, pp. 1217–1229, 2008.

[62] T. Elsayed, J. Lin, and D. W. Oard, “Pairwise Document Similarity in Large Collections with MapReduce,” *ACL*, no. June, pp. 265–268, 2008.

APPENDIX A – IDENTITY SERVER VERIFICATION PROCESS

A.1 – Identity Server Output

```
02/17/2017 22:49:03 -06:00 [WRN]
        (IdentityServer3.Core.Configuration.IdentityServerServiceFactory)
         AuthorizationCodeStore not configured - falling back to InMemory
02/17/2017 22:49:03 -06:00 [WRN]
        (IdentityServer3.Core.Configuration.IdentityServerServiceFactory)
         TokenHandleStore not configured - falling back to InMemory
02/17/2017 22:49:03 -06:00 [WRN]
        (IdentityServer3.Core.Configuration.IdentityServerServiceFactory)
        ConsentStore not configured - falling back to InMemory
02/17/2017 22:49:03 -06:00 [WRN]
        (IdentityServer3.Core.Configuration.IdentityServerServiceFactory)
        RefreshTokenStore not configured - falling back to InMemory

Server listening at https://localhost:44333/core. Press enter to stop

02/17/2017 22:49:33 -06:00 [INF]
        (IdentityServer3.Core.Endpoints.TokenEndpointController)
        Start token request
02/17/2017 22:49:33 -06:00 [DBG]
        (IdentityServer3.Core.Validation.ClientSecretValidator)
        Start client validation
02/17/2017 22:49:33 -06:00 [DBG]
        (IdentityServer3.Core.Validation.BasicAuthenticationSecretParser)
        Start parsing Basic Authentication secret
02/17/2017 22:49:33 -06:00 [DBG]
        (IdentityServer3.Core.Validation.SecretParser)
         Parser found secret: BasicAuthenticationSecretParser
02/17/2017 22:49:33 -06:00 [INF]
        (IdentityServer3.Core.Validation.SecretParser)
        Secret id found: ro.client
02/17/2017 22:49:33 -06:00 [DBG]
        (IdentityServer3.Core.Validation.SecretValidator)
        Secret validator success: HashedSharedSecretValidator
02/17/2017 22:49:33 -06:00 [INF]
        (IdentityServer3.Core.Validation.ClientSecretValidator)
        Client validation success
02/17/2017 22:49:33 -06:00 [INF]
```

(IdentityServer3.Core.Validation.TokenRequestValidator)
Start token request validation
02/17/2017 22:49:33 -06:00 [INF]
(IdentityServer3.Core.Validation.TokenRequestValidator)
Start password token request validation
02/17/2017 22:49:33 -06:00 [INF]
(IdentityServer3.Core.Validation.TokenRequestValidator)
Password token request validation success.
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Validation.TokenRequestValidator)
Token request validation success

```
 {
  "ClientId": "ro.client",
  "ClientName": "Resource Owner Flow Client",
  "GrantType": "password",
  "Scopes": "email openid roles",
  "UserName": "trustmarktest2@gmail.com",
  "AuthenticationContextReferenceClasses": [],
  "Raw": {
"grant_type": "password",
"username": "trustmarktest2@gmail.com",
"password": "******",
"scope": "roles openid email"
  }
}
```


02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.ResponseHandling.TokenResponseGenerator)
Creating token response
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.ResponseHandling.TokenResponseGenerator)
Processing token request
02/17/2017 22:49:34 -06:00 [DBG]
(IdentityServer3.Core.Services.Default.DefaultTokenService)
Creating access token
02/17/2017 22:49:34 -06:00 [DBG]
(IdentityServer3.Core.Services.Default.DefaultTokenService)
Creating JWT access token
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Endpoints.TokenEndpointController)
End token request
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Results.TokenResult)
Returning token response.
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Endpoints.UserInfoEndpointController)

Start userinfo request
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Endpoints.UserInfoEndpointController)
Token found: AuthorizationHeader
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Validation.TokenValidator)
Start access token validation
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Validation.TokenValidator)
Token validation success

```
{
  "ValidateLifetime": true,
  "AccessTokenType": "Jwt",
  "ExpectedScope": "openid",
  "Claims": {
"iss": "https://localhost:44333/core",
"aud": "https://localhost:44333/core/resources",
"exp": "1487396974",
"nbf": "1487393374",
"client_id": "ro.client",
"scope": [
  "email",
  "openid",
  "roles"
],
"sub": "153db49a-48d7-4d16-bc35-3c09d0bd3734",
"auth_time": "1487393373",
"idp": "idsrv",
"amr": "password"
  }
}
```

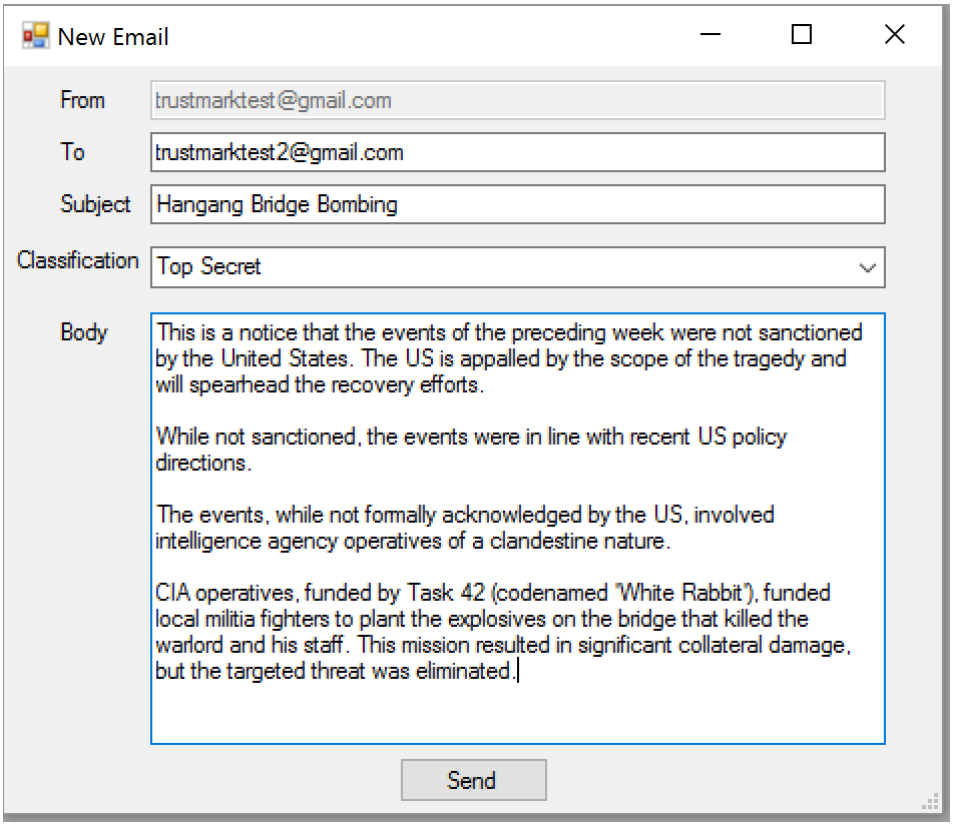

02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.ResponseHandling.UserInfoResponseGenerator)
Creating userinfo response
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.ResponseHandling.UserInfoResponseGenerator)
Scopes in access token: email openid roles
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.ResponseHandling.UserInfoResponseGenerator)
Requested claim types: email email_verified sub role
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.ResponseHandling.UserInfoResponseGenerator)
Profile service returned to the following claim types: sub email email_verified role role role
02/17/2017 22:49:34 -06:00 [INF]

(IdentityServer3.Core.Endpoints.UserInfoEndpointController)
End userinfo request
02/17/2017 22:49:34 -06:00 [INF]
(IdentityServer3.Core.Results.UserInfoResult)
Returning userinfo response.

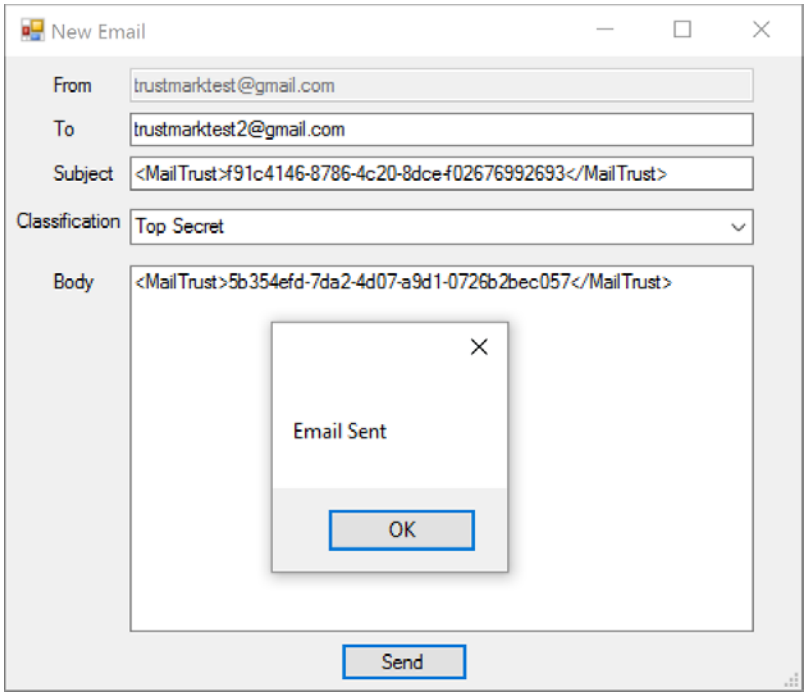
APPENDIX B – EMAIL EXAMPLE EMAIL CLIENT

B.1 - Example MailTrust Client

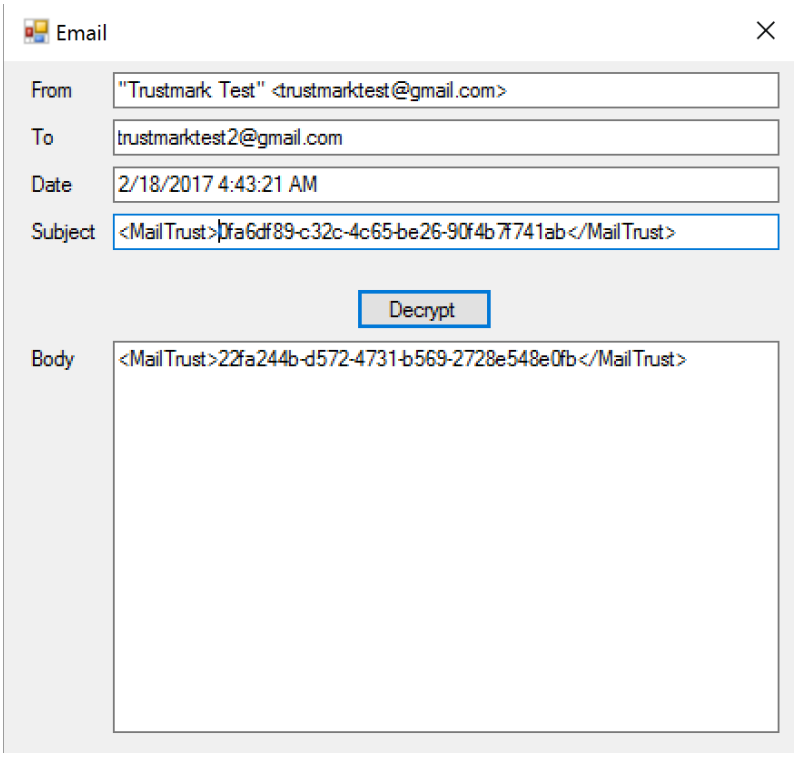
B.2 – Creating a MailTrust Email



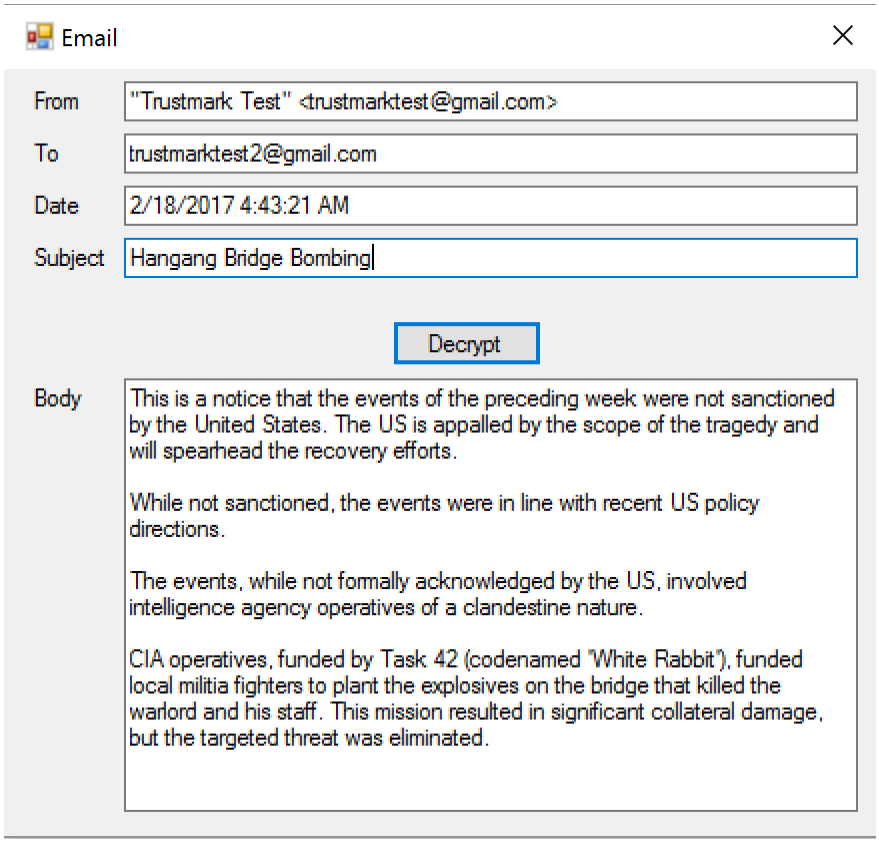
B.3 – Message After Being Sent



B.4 – Message Before Decryption



B.5 – Message After Decryption



APPENDIX C – IDENTITY SERVER USER SETUP

C.1 - Users.cs

```
using IdentityServer3.Core;
using IdentityServer3.Core.Services.InMemory;
using System.Collections.Generic;
using System.Security.Claims;

namespace IdentityServer3.Host.Config
{
static class Users
{
public static List<InMemoryUser> Get()
{
var users = new List<InMemoryUser>
{

new InMemoryUser{Subject = "54bd5ccf-9b6d-4399-878f-37585512f1c4",
Username = "trustmarktest@gmail.com", Password = "testpass",
Claims = new Claim[]
{
new Claim("Attribute", "test"),
new Claim(Constants.ClaimTypes.Name, "Trustmark Test"),
new Claim(Constants.ClaimTypes.GivenName, "Trustmark"),
new Claim(Constants.ClaimTypes.FamilyName, "Test"),
new Claim(Constants.ClaimTypes.Email, "trustmarktest@gmail.com"),
new Claim(Constants.ClaimTypes.EmailVerified, "true",
ClaimValueTypes.Boolean),
new Claim(Constants.ClaimTypes.Role, "Unclassified"),
new Claim(Constants.ClaimTypes.Role, "Secret"),
new Claim(Constants.ClaimTypes.WebSite, "http://trustmarktest.com"),
new Claim(Constants.ClaimTypes.Address, @"{ ""street_address"": ""One
Hacker Way"", ""locality"": ""Heidelberg"", ""postal_code"": 69118,
""country"": ""United States"" }", Constants.ClaimValueTypes.Json),
```

```
new Claim(Constants.ClaimTypes.Secret,
"MIIBITANBgkqhkiG9w0BAQEFAAOCAQ4AMIIBCQKCAQB596bHdOHfyKvofejrcWFdnHZLk
ISgIt/02DYTjx34HshfQ1mkW8GoSB9hGKTKSgf/r3WwZfItZkwidEIlSdbHrWeF5jQsQOI
ZtV5Ywe/cHnJSShVXTdZLB34nmXooOMyZk2ZcxCyJ7hN/jjz0Eq7PD0Sek2gfdF0bHcIgM
53XEKbb/PxQFvFWmk2hkwyfwYNNvgZOj1QGyav52wNr1odrvYiop2ypFF8YmuDFNx1EU5f
st96Jx7GPYNyPoycAaIdWZkTwwoT260NSQMkA2opTL2aEV0HhmnE1I2k25sRlqDHPYIClb
CEXAG3EAgyAXutVrsg6+wktxAdHDyK8qcEPAgMBAAE="),
}
},

new InMemoryUser{Subject = "153db49a-48d7-4d16-bc35-3c09d0bd3734",
Username = "trustmarktest2@gmail.com", Password = "testpass",
Claims = new Claim[]
{
new Claim(Constants.ClaimTypes.Name, "Trustmark Test2"),
new Claim(Constants.ClaimTypes.GivenName, "Trustmark"),
new Claim(Constants.ClaimTypes.FamilyName, "Test2"),
new Claim(Constants.ClaimTypes.Email, "trustmarktest2@gmail.com"),
new Claim(Constants.ClaimTypes.EmailVerified, "true",
ClaimValueTypes.Boolean),
new Claim(Constants.ClaimTypes.Role, "Unclassified"),
new Claim(Constants.ClaimTypes.Role, "Classified"),
new Claim(Constants.ClaimTypes.Role, "Secret"),
new Claim(Constants.ClaimTypes.WebSite, "http://trustmarktest2.com"),
new Claim(Constants.ClaimTypes.Address, @"{ ""street_address"": ""One
Hacker Way"", ""locality"": ""Heidelberg"", ""postal_code"": 69118,
""country"": ""Germany"" }", Constants.ClaimValueTypes.Json),
new Claim(Constants.ClaimTypes.Secret,
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtFSrRGGPrM79AXNK8euZT2ArB
MBRHiWP5ptXLIvjn0/FFiQmmsy3XL7yiSwvv+jN2zzdh2VbMZNFE9XHVGn79M7p8iSHehb
VO7GLG2xQioKhjyzC05rl+zbCRJCGgA5GxB2NwopZY0a3+E0U/q0j+MN4KG5HM+Qehh3hO
ZKYPpHHQlfmqaKiVVYsid1PQuB8Hq+wGz1qYcKJclbTEo5jQDYACro89E70cw7ct0F1tI+
+YuCwLqpBpwfJIVXJF9GGnboQrbqoMeuDoMmIubWh8cUQW4T0ux/EjGZbTzMo9fe/sdvmu
C5WWT9H/aqI7TYq5iYcR5/nPHJAoAYcdlObHwIDAQAB"),
}
},

new InMemoryUser { Subject = "50d43b2d-f344-495d-b7f3-b5aaade85ec5",
Username = "trustmarktest3@gmail.com", Password = "testpass",
Claims = new Claim[]
```

```
{
new Claim(Constants.ClaimTypes.Name, "Trustmark Test3"),
new Claim(Constants.ClaimTypes.GivenName, "Trustmark"),
new Claim(Constants.ClaimTypes.FamilyName, "Test3"),
new Claim(Constants.ClaimTypes.Email, "trustmarktest3@gmail.com"),
new Claim(Constants.ClaimTypes.EmailVerified, "true",
ClaimValueTypes.Boolean),
new Claim(Constants.ClaimTypes.Role, "Unclassified"),
new Claim(Constants.ClaimTypes.Role, "Classified"),
new Claim(Constants.ClaimTypes.Role, "Secret"),
new Claim(Constants.ClaimTypes.Role, "Top Secret"),
new Claim(Constants.ClaimTypes.WebSite, "http://bob.com"),
new Claim(Constants.ClaimTypes.Address, @"{ ""street_address"": ""One
Hacker Way"", ""locality"": ""Heidelberg"", ""postal_code"": 69118,
""country"": ""United Kingdom"" }", Constants.ClaimValueTypes.Json),
new Claim(Constants.ClaimTypes.Secret,
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjtjQZkKRae4iEshm7bNuZn8sy
7L/42qtIWedNEoCMk8km7YobxtnNwa/zGk7JjqjY9meyuTqhwPCY35Wv02W4qQvqsk8bpE
LzgmOk5iFSZWbcfPlMNHdfT+0sccKF1P9FQML/Ixdb+qgwqlp3JaCOlHTUTWQfY/h+m2A/
W3T/LXXLtU/doxUWcS7W3X1/juVG3EhooaXvzKu5mKpBNwmHqOsawa6UeuOh0CSty/uygN
etI2v2GDwpyaqrscC9VDX7Ye6o7yaSDGIwsmChevRCFh7XDGY5Z9f5EJG7Hj+yrrYG3fMS
LhM+HeK2lrlRtp8ZxCacyvRt4k/7CyNL0WBpwIDAQAB"),
}
},

new InMemoryUser{Subject = "ac26c87f-2139-4bdd-be67-1cf8947a32d7",
Username = "trustmarktest4@gmail.com", Password = "testpass",
Claims = new Claim[]
{
new Claim(Constants.ClaimTypes.Name, "Trustmark Test4"),
new Claim(Constants.ClaimTypes.GivenName, "Trustmark"),
new Claim(Constants.ClaimTypes.FamilyName, "Test4"),
new Claim(Constants.ClaimTypes.Email, "trustmarktest4@gmail.com"),
new Claim(Constants.ClaimTypes.EmailVerified, "true",
ClaimValueTypes.Boolean),
new Claim(Constants.ClaimTypes.Role, "Unclassified"),
new Claim(Constants.ClaimTypes.Role, "Classified"),
new Claim(Constants.ClaimTypes.WebSite, "http://bob.com"),
```

```
new Claim(Constants.ClaimTypes.Address, @"{ ""street_address"": ""One
Hacker Way"", ""locality"": ""Heidelberg"", ""postal_code"": 69118,
""country"": ""China"" }", Constants.ClaimValueTypes.Json),
new Claim(Constants.ClaimTypes.Secret,
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAlxc/mK0zxPd94aodvdCdXqTNu
PPB7BjxXq8VI1RkEgVZFRvtL/xdHYUlNADkOVFXuupUmMrSSqVAb+JJPgxfmbJD+4pwdA9
kKehI689YKL8HANpEg6Dgn4fo90uKvA3FP5kd3ZvHmQeyHYZEaxKH2vggClJrZABCs4wRA
xJyQfsgUyePB/XFjx43z0c2DSRC/uIX3XKcd1I4+krKh93IUY+kKHJg/3H8+4lNjOHe9Jx
JcaSk0O82DxyIKVLsF8DicncRCE/JBHLUcdlJQFi3u03QZQKjlAKUtM6rC2WNnglCzqEro
I/dXGugO6lqcNIrT282PeKNwTvl3UduFfDH9QIDAQAB"),
}
},
};

return users;
}
}
```

APPENDIX D – SCOPES, CLIENTS, & RELYING PARTIES

D.1 - Scopes.cs

```
using IdentityServer3.Core;
using IdentityServer3.Core.Models;
using System.Collections.Generic;

namespace IdentityServer3.Host.Config
{
public class Scopes
{
public static IEnumerable<Scope> Get()
{
return new[]
{
////////////////////////
// identity scopes
////////////////////////

StandardScopes.OpenId,
StandardScopes.Profile,
StandardScopes.Email,
StandardScopes.Address,
StandardScopes.OfflineAccess,
StandardScopes.RolesAlwaysInclude,
StandardScopes.Roles,


////////////////////////
// resource scopes
////////////////////////

new Scope
```

```
{
Name = "read",
DisplayName = "Read data",
Type = ScopeType.Resource,
Emphasize = false,

ScopeSecrets = new List<Secret>
{
new Secret("secret".Sha256())
}
},
new Scope
{
Name = "write",
DisplayName = "Write data",
Type = ScopeType.Resource,
Emphasize = true,

ScopeSecrets = new List<Secret>
{
new Secret("secret".Sha256())
}
},
new Scope
{
Name = "idmgr",
DisplayName = "IdentityManager",
Type = ScopeType.Resource,
Emphasize = true,
ShowInDiscoveryDocument = false,

Claims = new List<ScopeClaim>
{
new ScopeClaim(Constants.ClaimTypes.Name),
new ScopeClaim(Constants.ClaimTypes.Role)
}
}
};
}
}
```

```
}
```

D.2 - Clients.cs

```
using IdentityServer3.Core;
using IdentityServer3.Core.Models;
using System.Collections.Generic;
using System.Security.Claims;

namespace IdentityServer3.Host.Config
{
public class Clients
{
public static List<Client> Get()
{
return new List<Client>
{
new Client
{
ClientName = "Resource Owner Flow Client",
ClientId = "ro.client",
Flow = Flows.ResourceOwner,

ClientSecrets = new List<Secret>
{
new Secret("secret".Sha256())
},

AllowedScopes = new List<string>
{
"openid",
"email",
"read",
"write",
Constants.StandardScopes.Roles,
"offline_access"
},

// used by JS resource owner sample
AllowedCorsOrigins = new List<string>
```

```
{
"http://localhost:13048"
},

AccessTokenType = AccessTokenType.Jwt,
AccessTokenLifetime = 3600,

// refresh token settings
AbsoluteRefreshTokenLifetime = 86400,
SlidingRefreshTokenLifetime = 43200,
RefreshTokenUsage = TokenUsage.OneTimeOnly,
RefreshTokenExpiration = TokenExpiration.Sliding
},


};
}
}

}
```

D.3 - RelyingParties.cs

```
using System.Collections.Generic;
using System.Security.Claims;
using IdentityServer3.WsFederation.Models;
using IdentityModel.Constants;

namespace SelfHost.Config
{
public class RelyingParties
{
public static IEnumerable<RelyingParty> Get()
{
return new List<RelyingParty>
{
new RelyingParty
{
Realm = "urn:testrp",
Name = "Test RP",
```

```
Enabled = true,
ReplyUrl = "https://web.local/idsrvrp/",
TokenType = TokenTypes.Saml2TokenProfile11,
TokenLifeTime = 1,

ClaimMappings = new Dictionary<string,string>
{
{ "sub", ClaimTypes.NameIdentifier },
{ "given_name", ClaimTypes.Name },
{ "email", ClaimTypes.Email }
}
},
new RelyingParty
{
Realm = "urn:owinrp",
Enabled = true,
ReplyUrl = "http://localhost:10313/",
TokenType = TokenTypes.Saml2TokenProfile11,
TokenLifeTime = 1,

ClaimMappings = new Dictionary<string, string>
{
{ "sub", ClaimTypes.NameIdentifier },
{ "name", ClaimTypes.Name },
{ "given_name", ClaimTypes.GivenName },
{ "email", ClaimTypes.Email }
}
}
};
}
}
}
```

APPENDIX E – ENCRYPTION & DECRYPTION METHODS

E.1 - StringCypher.cs – Original Source [56]

```
using System;
using System.Text;
using System.Security.Cryptography;
using System.IO;
using System.Linq;

namespace IDPServices
{
public static class StringCipher
{
// This constant is used to determine the keysize of the encryption
algorithm in bits.
// We divide this by 8 within the code below to get the equivalent
number of bytes.
private const int Keysize = 256;

// This constant determines the number of iterations for the password
bytes generation function.
private const int DerivationIterations = 100000;

public static string Encrypt(string plainText, string passPhrase)
{
// Salt and IV is randomly generated each time, but is preprended to
encrypted cipher text
// so that the same Salt and IV values can be used when decrypting.
var saltStringBytes = Generate256BitsOfRandomEntropy();
var ivStringBytes = Generate256BitsOfRandomEntropy();
var plainTextBytes = Encoding.UTF8.GetBytes(plainText);

using (var password = new Rfc2898DeriveBytes(passPhrase,
saltStringBytes, DerivationIterations))
```

```
{

DateTime start = DateTime.Now;
var keyBytes = password.GetBytes(Keysize / 8);
TimeSpan ts = DateTime.Now - start;
using (var symmetricKey = new RijndaelManaged())
{
symmetricKey.BlockSize = 256;
symmetricKey.Mode = CipherMode.CBC;
symmetricKey.Padding = PaddingMode.PKCS7;
using (var encryptor = symmetricKey.CreateEncryptor(keyBytes,
ivStringBytes))
{
using (var memoryStream = new MemoryStream())
{
using (var cryptoStream = new CryptoStream(memoryStream, encryptor,
CryptoStreamMode.Write))
{
cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);
cryptoStream.FlushFinalBlock();
// Create the final bytes as a concatenation of the random salt bytes,
the random iv bytes and the cipher bytes.
var cipherTextBytes = saltStringBytes;
cipherTextBytes = cipherTextBytes.Concat(ivStringBytes).ToArray();
cipherTextBytes =
cipherTextBytes.Concat(memoryStream.ToArray()).ToArray();
memoryStream.Close();
cryptoStream.Close();
return Convert.ToBase64String(cipherTextBytes);
}
}
}
}
}
}

public static string Decrypt(string cipherText, string passPhrase)
{
// Get the complete stream of bytes that represent:
// [32 bytes of Salt] + [32 bytes of IV] + [n bytes of CipherText]
```

```
var cipherTextBytesWithSaltAndIv =
Convert.FromBase64String(cipherText);
// Get the saltbytes by extracting the first 32 bytes from the
supplied cipherText bytes.
var saltStringBytes = cipherTextBytesWithSaltAndIv.Take(Keysize /
8).ToArray();
// Get the IV bytes by extracting the next 32 bytes from the supplied
cipherText bytes.
var ivStringBytes = cipherTextBytesWithSaltAndIv.Skip(Keysize /
8).Take(Keysize / 8).ToArray();
// Get the actual cipher text bytes by removing the first 64 bytes
from the cipherText string.
var cipherTextBytes = cipherTextBytesWithSaltAndIv.Skip((Keysize / 8)
* 2).Take(cipherTextBytesWithSaltAndIv.Length - ((Keysize / 8) *
2)).ToArray();

using (var password = new Rfc2898DeriveBytes(passPhrase,
saltStringBytes, DerivationIterations))
{
var keyBytes = password.GetBytes(Keysize / 8);
using (var symmetricKey = new RijndaelManaged())
{
symmetricKey.BlockSize = 256;
symmetricKey.Mode = CipherMode.CBC;
symmetricKey.Padding = PaddingMode.PKCS7;
using (var decryptor = symmetricKey.CreateDecryptor(keyBytes,
ivStringBytes))
{
using (var memoryStream = new MemoryStream(cipherTextBytes))
{
using (var cryptoStream = new CryptoStream(memoryStream, decryptor,
CryptoStreamMode.Read))
{
var plainTextBytes = new byte[cipherTextBytes.Length];
var decryptedByteCount = cryptoStream.Read(plainTextBytes, 0,
plainTextBytes.Length);
memoryStream.Close();
cryptoStream.Close();
return Encoding.UTF8.GetString(plainTextBytes, 0, decryptedByteCount);
}
```

```
}
}
}
}
}

private static byte[] Generate256BitsOfRandomEntropy()
{
var randomBytes = new byte[32]; // 32 Bytes will give us 256 bits.
using (var rngCsp = new RNGCryptoServiceProvider())
{
// Fill the array with cryptographically secure random bytes.
rngCsp.GetBytes(randomBytes);
}
return randomBytes;
}
}

}
```

APPENDIX F – IRB APPROVAL

F.1 – IRB Approval Letter

THE UNIVERSITY OF ALABAMA | *Office of the Vice President for* Research & Economic Development
Office for Research Compliance

February 21, 2017

Matthew Hudnall
Deputy Director
Center for Advanced Public Safety
The University of Alabama
Box 8702899

Re: IRB Requirement for Computer Science Dissertation

Mr. Hudnall:

This letter comes as a response to your communication received February 20, 2017. According to the Office for Human Research Protection (OHRP) under policy 45 CFR 46.101 the proposed work is not human subjects research.

Because the work is not considered human subjects research, it does not require IRB approval and is therefore excluded from review by the IRB.

If you have any questions or if I can be of further assistance please do not hesitate to contact me.

Sincerely,

Carpantato T. Myles, MSM, CIM, CIP
Director & Research Compliance Officer
Office of Research Compliance

358 Rose Administration Building | Box 870127 | Tuscaloosa, AL 35487-0127
205-348-8461 | Fax 205-348-7189 | Toll Free 1-877-820-3066