# Mimesis Aegis: A Mimicry Privacy Shield

## A System's Approach to Data Privacy on Public Cloud

Billy Lau, Simon Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332*

{billy, pchung, csong84, yeongjin.jang, wenke, sasha.boldyreva}@cc.gatech.edu

## Abstract

Users are increasingly storing, accessing, and exchanging data through public cloud services such as those provided by Google, Facebook, Apple, and Microsoft. Although users may want to have faith in cloud providers to provide good security protection, the Snowden exposé is the latest reminder of the reality we live in: the confidentiality of any data in public clouds can be violated, and consequently, while the providers may not be "doing evil", we can not and should not trust them with data confidentiality.

To better protect the privacy of user data stored on the cloud, in this paper we propose a privacy-preserving system called *Mimesis Aegis (M-Aegis)* that is suitable for mobile platforms. M-Aegis is a new approach to user data privacy that not only provides isolation but also preserves user experience, through the creation of a conceptual layer called *Layer 7.5 (L-7.5)*, which is interposed between the application (Layer 7) and the user (Layer 8). This approach allows M-Aegis to implement a true end-to-end encryption of user data with three goals in mind: 1) complete data and logic isolation from untrusted entities; 2) the preservation of original user experience with target apps; and 3) applicable to a large number of apps and resilient to updates.

In order to preserve the exact application workflow and look-and-feel, M-Aegis uses L-7.5 to put a transparent window on top of existing application's GUIs to both intercept plaintext user input before transforming the input (if necessary) and feeding it to the underlying app, and reverse-transform (if necessary) the output data from the app before displaying the plaintext data to the user. This technique allows M-Aegis to transparently integrate with most cloud services without hindering usability and without the need for reverse-engineering. We implemented a prototype of M-Aegis on Android and show that it can support a number of popular cloud services, e.g. Gmail, Facebook Messenger, WhatsApp, etc.

Our performance evaluation and user study show that users incur minimal overhead in adopting M-Aegis on Android: imperceptible encryption/decryption latency and very low and adjustable false positive rate when searching over encrypted data.

## 1 Introduction

As a result of the growth in cloud computing and mobile technology, today we witness a continuously increasing number of users who utilize mobile devices [2] to interact with public cloud services (PCS) (e.g. Gmail, Outlook, and WhatsApp) as an essential part of their daily lives. Generally, while the user's connectivity to the Internet is improved, the problem of preserving data privacy in the interaction with PCS is yet unsolved. In fact, news about the US government's alleged surveillance programs reminds everybody about a very unsatisfactory status quo: while PCS are essentially part-of-life, the default way of utilizing them exposes users to privacy breaches, because it implicitly requires the users to trust the PCS providers with the confidentiality of their data, and thus their privacy; but such trust truly is *unjustified*, if not misplaced. Incidents that demonstrate the breach of this trust are easy to come by: 1) PCS providers are bounded by law to share their users' data with surveillance agencies [14], 2) it is the business model of the PCS providers to go through their users' data and share it with third parties [11, 22, 25, 44], 3) operator errors [38] can result in unintended data access, and 4) data servers can be compromised by attackers [51].

To alter this undesirable status quo, solutions should be built based on an updated trust model of everyday communication that better reflects the reality of threats mentioned earlier. In particular, new solutions must first assume PCS providers to be untrusted. This implies that all other entities that are controlled by the PCS providers, including the apps that users installed to engage with the PCS, must also be assumed untrusted.

Although there are a plethora of apps available today

that comes in various combinations of look-and-feel and features, we observed that for a large class of these apps that provides text communication services (e.g. email or private/group messaging categories), users can still enjoy the same quality of service[1] without needing to reveal their plaintext data to the PCS providers. PCS providers are essentially message routers that can function normally without needing to know the content of the messages being delivered, analogous to postmen delivering letters without needing to learn the actual content of the letters.

Hence, in theory, applying end-to-end encryption (E2EE) without assuming trust in the PCS providers seems to solve the problem. However, in practice, the direct application of E2EE solutions onto the mobile device environment is more challenging than originally thought [68, 62]. A good solution must present clear advantages to the entire mobile security ecosystem, in particular accounting for these factors: the users' ease-of-use, hence acceptability and adoptability; the developers' efforts to maintain support, and the feasibility and deployability of solution on the mobile system. From this analysis, we formulate three key challenges that must be addressed coherently:

1. For a solution to be secure, it must be properly isolated from untrusted entities. It is obvious that E2EE cannot protect data confidentiality if the plaintext data or even the encryption key can be compromised by architectures that risk exposing these values. Traditional solutions like PGP [15] and newer solutions like Gibberbot [5], TextSecure [12], and SafeSlinger [45] provide good isolation property, but force users to use custom apps, which can cause usability problems (refer to (2)). Solutions that repackage/rewrite existing apps to introduce additional security checks [71, 30] do not have this property (further discussed in Sect. 2.3). Solutions in the form of browser plugins/extensions also do not have this property (further discussed in Sect. 2.2), and they generally do not fit into the mobile security landscape because besides the fact that mobile browsers do not support extensions [7], mobile device users simply do not favor using mobile browsers [31] to access PCS. Therefore, we rule out conventional browser-extension/plugin-based solutions.
2. For a solution to be adoptable, it must preserve the user experience. We posit that users will not accept solutions that require them to switch between different apps to perform their daily tasks. Therefore, simply porting solutions like PGP to mobile platform would not work, because besides forcing users to use a separate and custom app, it is impossible to recreate the richness and unique user experience of all existing email apps offered by various PCS providers today. In the context of mobile devices, PCS nowadays are competing for market share not only by offering more reliable infrastructure to facilitate user communication, but also by offering a better user experience [16, 61]. Ultimately, users will choose apps that they feel most comfortable with. To reduce interference with the user's interaction with the app of their choice, security solutions must be retrofittable to existing apps. Solutions that repackage/rewrite existing apps have this criterion.
3. For a solution to be sustainable, it must be easy to maintain and scalable: the solution must be sufficiently general-purpose, require minimal efforts to support new apps and withstand app updates. In the past, email was one of the very few means of communication. Protecting it is relatively straightforward because email protocols (e.g. POP and IMAP) are well defined. Custom privacy-preserving apps can therefore be built to serve this need. However, with the plethora of PCS that are becoming indispensable in a user's everyday life today, a good solution should also be able to integrate security features to apps without requiring reverse engineering the apps' logic and/or network protocols, which are largely undocumented and possibly proprietary (e.g. Skype, WhatsApp, etc.).

In this paper, we introduce *Mimesis Aegis (M-Aegis)*, a privacy-preserving system that "mimics" the look and feel of existing apps to preserve their user experience and workflow on mobile devices, without changing the underlying OS or modifying/repackaging existing apps. M-Aegis achieves the design goals by operating at a conceptual layer we call *Layer 7.5 (L-7.5)* that is positioned above the existing application layer (OSI Layer 7 [8]), and interacts directly with the user (popularly labeled as Layer 8 [19, 4]).

From a system's perspective, L-7.5 is a transparent window in an isolated process that interposes itself between Layer 7 and 8. The interconnectivity between these layers is achieved using the accessibility framework, which is available as an essential feature on modern operating systems (OS). Note that utilizing accessibility features for unorthodox purposes have been proposed by prior work [59, 52] that achieve different goals. L-7.5 extracts the GUI information of the app below it through the OS's user interface automation/accessibility (UIA) library. Using this information, M-Aegis is then able to "proxy" user input by rendering its own GUI (with a different color as visual cue) and subsequently handle those input (e.g. to process plaintext data or intercept user button click). Using the same UIA library,

[1] the apps' functionalities and user experience are preserved

L-7.5 can also programmatically interact with various UI components of the app below on behalf of the user (refer to Sect. 3.3.2 for more details). Since major software vendors today have pledged their commitment towards continuous support and enhancement of accessibility interface for developers [9, 20, 6, 1], our UIA-based technique is applicable and sustainable on all major platforms.

From a security design's perspective, M-Aegis provides two privacy guarantees during a user's interaction with a target app: 1) all input from the user first goes to L-7.5 (and be optionally processed) before being passed to the app. This means that confidential data and user intent can be fully captured; and 2) all output from the app must go through L-7.5 (and be optionally processed) before being displayed to the user.

From a developer's perspective, accessing and interacting with a target app's UI components at L-7.5 is very similar to that of manipulating the DOM tree of a web app using Javascript. However, one major difference is that DOM tree manipulation only works for browsers, but UIA works for all apps on a platform. To track the GUI of an app, M-Aegis relies on resource id names available through the UIA library. Therefore, M-Aegis is resilient to updates that change the look and feel of the app (e.g. GUI position or color). It only breaks when the resource id names are changed, which, through empirical evidences, are very rare occasions. Even if such a case happens, a very minimal effort is required to rediscover the resource id names and remap them to the logic in M-Aegis. From our experience so far, our solution does not require developer attention across a few minor app updates.

From a user's perspective, M-Aegis is visible as an always-on-top button. When it is turned on, users will perceive that they are interacting with the original app in plaintext mode. The only difference is the GUI of the original app will appear in a different color to indicate that protection is activated. This means that subtle features that contribute towards the entire user experience such as spellcheck and in-app navigation are also preserved. However, despite user perception, the original app actually *never* receives the plaintext data. Figure 1 gives a high level idea of how M-Aegis creates an L-7.5 to protect user's data privacy when interacting with Gmail.

For users who would like to protect their email communications, they will also be concerned if encryption will affect the ability to search, because it is very important to improve user productivity [67]. For this purpose, we designed and incorporated a new searchable encryption scheme named *easily-deployable efficiently-searchable symmetric encryption scheme* (EDESE) into M-Aegis that allows search over encrypted content without any server-side modification. We briefly discuss the design considerations and security concerns involved in supporting this functionality in Sect 3.3.4.

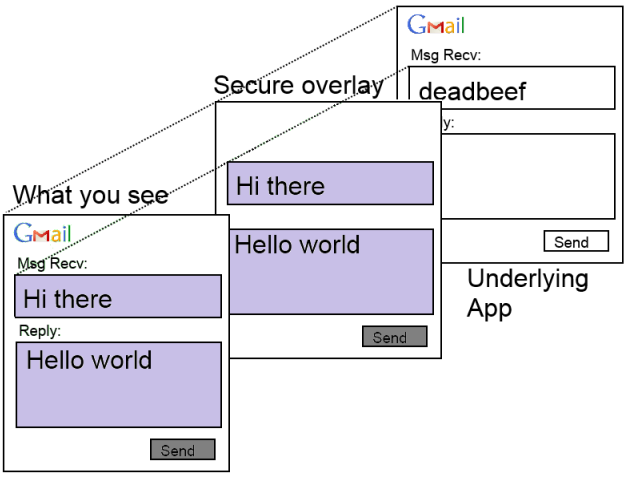


Figure 1: This diagram shows how M-Aegis uses L-7.5 to transparently reverse-transform the message "deadbeef" into "Hi there", and also allow user to enter their plaintext message "Hello world" into M-Aegis's text box. To the user, the GUI looks exactly the same as the original app. When the user decides to send it out, the "Hello world" message will be transformed and relayed to the underlying app.

As a proof of concept, we implemented a prototype M-Aegis on Android that protects user data when interfacing with text-based PCS. M-Aegis supports email apps like Gmail and messenger apps like Google Hangout, WhatsApp, and Facebook Chat. It protects data privacy by implementing a true E2EE without trusting PCS apps by operating at L-7.5 while preserving the user experience and workflow of the target apps. We also implemented a version of M-Aegis on desktop to demonstrate the generality of our approach. Our initial performance evaluation and user study show that users incur minimal overhead in adopting M-Aegis on Android: imperceptible encryption/decryption latency and very low and adjustable false positive rate when searching over encrypted data.

In summary, these are the major contributions of this work:

- We introduced Layer 7.5 (L-7.5), a conceptual layer that directly interacts with users on top of existing apps. This is a novel system approach that provides seemingly contrasting features: transparent interaction with a target app and strong isolation from the target app.
- We designed and built Mimesis Aegis based on the concept of L-7.5, a system that preserves user's pri-

vacy while interacting with PCS by protecting data confidentiality. Essential functionalities of existing apps, especially search (even over encrypted data), are also supported without any server-side modification.
- We implemented two prototypes of Mimesis Aegis, one on Android and the other on Windows, with support for various popular public cloud services, including Gmail, Facebook Messenger, Google Hangout, WhatsApp, and Viber.
- We designed and conducted a user study that demonstrated the acceptability of our solution.

The rest of the paper is structured as follows. Section 2 compares our work from related work. Section 3 discusses the threat model and the design of M-Aegis. Section 4 presents the implementation of M-Aegis and the challenges we solved during the process. Section 5 presents performance evaluations and user study of the acceptability of M-Aegis on Android. Section 6 discusses limitations of our work and answers some common questions that readers may have about our system. Section 7 discusses future work and concludes our work.

## 2 Related Work

Since M-Aegis is designed to achieve the three design goals described earlier while seamlessly integrating end-to-end encryption into user's communication, we discuss how well existing works achieve some of these goals and how they differ from our work. As far as we know, there is no existing work that achieves all the three necessary design goals.

### 2.1 Standalone Solutions

There are many standalone solutions that aim to protect user's data confidentiality. Solutions like PGP [15] (including S/MIME [41]), Gibberbot [5], TextSecure [12], SafeSlinger [45], and FlyByNight [58] provides secure messaging and/or file transfer through the encryption of user data. These solutions provide good isolation property from untrusted entities. However, since they are designed as standalone custom apps, they do not preserve user experience, requiring users to adapt to new workflow on a custom app. More importantly, these solutions are not retrofittable to existing apps on the mobile platform.

Like M-Aegis, Cryptons [40] introduced a similarly strong notion of isolation through its custom abstractions. However, Cryptons assumes a completely different threat model in that it assumes PCS to be trusted and thus requires both server and client (app) modifications, of which our work does not. Essentially, Cryptons focuses on a problem that is different from M-Aegis. For example, Cryptons definitely could not protect a user's communication using existing messaging apps while assuming the provider to be untrusted. We also argue that it is non-trivial to modify Cryptons to achieve the three design goals we mentioned in Sect. 1.

### 2.2 Browser Plugin/Extention Solutions

Other solutions that focus on protecting user privacy include Cryptocat [3], Scramble! [25], TrustSplit's [44], NOYB (None of Your Business) [50], and SafeButton [57]. Some of these assume different threat models, and achieve different goals. For example, NOYB protects user's Facebook profile data while SafeButton tries to keep the user's browsing history private. Most of these solutions try to be transparently integrated into user workflow. However, since these solutions are mostly based on browser extension/plugins, they are not applicable to the mobile platform other than having questionable data isolation models.

Additionally, Cryptocat and TrustSplit require new and/or independent service providers to support their functionalities. However, M-Aegis works with the existing service providers without assuming trust or requiring modification to the servers.

### 2.3 Repackaging/Rewriting Solutions

There is a category of work that repackages/rewrites an app's binary to introduce security references, such as Aurasium [71], Dr. Android [53], and others [30]. Our solution is similar to these approaches in that we can retrofit our solutions to existing apps and still preserve user experience, but is different in that M-Aegis' coverage is not limited to apps that do not contain native code. Also, repackaging-based approaches suffer from the problem that it will break app updates. In some cases, the security of such solutions can be circumvented because the isolation model is unclear, i.e. the untrusted code resides in the same address space as the reference monitor (e.g. Aurasium).

### 2.4 Orthogonal Work

Although our work focuses on user interaction on mobile platform with cloud providers, we assume a very different threat model than those that focus on more robust permission model infrastructure and those that focus on controlling/tracking information flow, such as TaintDroid [42] and Airbag [70]. These solutions require changes to the underlying app, framework or the OS, but M-Aegis does not.

Access Control Gadgets (ACG) [60] uses user input as permission granting intent to allow apps to access user owned resources. Although we made the same assumptions as ACG to capture authentic user input, ACG is designed for a different threat model and security goal than ours. Further, ACG requires a modified kernel but M-Aegis does not.

Persona [24] presents a completely isolated and new online social network that provides certain privacy and security guarantees to the users. While related, it differs from the goal of M-Aegis.

Frientegrity [47] and Gyrus [52] focus on different aspects of integrity protection of the user's data.

Tor [39] is well known for its capability to hide a user's IP address while browsing the Internet. However, it focuses on anonymity guarantees while M-Aegis focuses on data confidentiality guarantees.

Off-the-record messaging (OTR) [34] is a secure communication protocol that provides perfect forward secrecy and malleable encryption. While OTR can be implemented on M-Aegis using the same design architecture to provide these extra properties, it is currently not the focus of our work.

# 3 System Design

## 3.1 Design Goals

In this section, we formally reiterate the design goals that our system wants to achieve. We posit that a good solution must:

1. offer good security by applying strong isolation from untrusted entities (defined in Sect. 3.2).
2. preserve user experience by allowing user a transparent interaction with existing apps.
3. be easy to maintain and scale by devising a sufficiently general-purpose approach.

Above all, these goals must be satisfied within the unique set of constraints found in the mobile platform, including user experience, transparency, deployability, and adoptability factors.

## 3.2 Threat Model

### 3.2.1 In-Scope Threats

We begin with the scope of threats that M-Aegis is designed to protect against. In general, there are three parties that pose threats to the confidentiality of users' data exposed to public cloud through mobile devices. Therefore, we assume these parties to be untrusted in our threat model:

- Public cloud service (PCS) providers. Sensitive data stored in the public cloud can be compromised in several ways: 1) the PCS providers are compelled by the law [21] to provide access to user's sensitive data to law enforcers [14]; 2) the business model of the PCS providers provides strong incentive for them to share/sell the data with third parties [11, 22, 25, 44]; 3) PCS administrators who have access to the sensitive data may also compromise the data, either intentionally (e.g., Edward Snowden) [14] or not [38]; and 4) vulnerabilities of the PCS can be exploited by attackers to exfiltrate sensitive data [51].
- Client-side apps. Since client-side apps are developed by the PCS providers to allow user to access their services, it follows that these apps are considered untrusted too.
- Middle boxes between the PCS and the client-side app. Finally, sensitive data can also be compromised when it is transferred between the PCS and the client-side app. Incorrect protocol design/implementation may allow attackers to eavesdrop on plaintext data or perform man-in-the-middle attacks [43, 18, 13].

M-Aegis addresses the above threats through isolation by first creating L-7.5 of which it would handle and apply end-to-end encryption (E2EE) on user private data. We consider the following components as our trusted computing base (TCB): the hardware, the operating system (OS) and the framework that controls and mediates access to hardware. In the absence of physical input devices (e.g. mouse and keyboard) on mobile devices, we additionally trust the soft keyboard to not leak what user have typed using the soft keyboard to the untrusted apps. We rely on the TCB to correctly handle I/O for M-Aegis, and to provide the proper isolation between M-Aegis and the untrusted components.

Additionally, we also assume that all the components of M-Aegis, including L-7.5 that it creates, are trusted. The user is also considered trustworthy under our threat model in his intent. This means that he is trusted to turn on M-Aegis when he wants to protect the privacy of his data during his interaction with the PCS.

### 3.2.2 Out of Scope Threats

Our threat model does not consider the following types of attacks. First, M-Aegis only guarantees the confidentiality of the user's data, but not its availability. Therefore, attacks that deny access to the data (denial-of-service) either at the server or the client are beyond the scope of this work. Second, any attacks against our TCB are orthogonal to this work. Such attacks include, but not limited to malicious hardware [56], attacks against the

hardware [69], the OS [54], the platform [66] and privilege escalation attacks (e.g. unauthorized rooting of device). However, note that M-Aegis can be implemented on a design that anchors its trust on trusted hardware and hypervisor (e.g. Gyrus [52], Storage Capsules [33]) to minimize the attack surface against TCB. Third, M-Aegis is designed to prevent any direct flow of information from authorized user to aforementioned untrusted entities. Hence, leakages through all side-channels [65] are beyond the scope of this work.

Since the user is assumed to be trustworthy under our threat model to use M-Aegis correctly, M-Aegis does not protect user against social engineering based attacks. For example, phishing attacks to trick users into either turning off M-Aegis and/or entering sensitive information into unprotected UI are beyond the scope of our paper. Instead, M-Aegis deploys best-effort by coloring the UI components in L-7.5 differently from that of the app's UI.

The other limitations of M-Aegis, which are not security threats, are discussed in Sect. 6.2.

## 3.3 M-Aegis Architecture

M-Aegis is architected to fulfill all of the three design goals mentioned in Sect. 3.1. Providing strong isolation guarantees is the first one. To achieve this, M-Aegis is designed to execute in a separate process although it resides on the same operating system (OS) as the target client app (TCA). Besides memory isolation, the filesystem of M-Aegis is also shielded from other apps by the OS' app sandbox protection.

Note that our architecture is flexible in that should greater degree of isolation be desirable, a virtual machine based setup can be adopted to provide even stronger security guarantees. However, we do not consider such design at this time as it is currently unsuitable for mobile platform, and the adoption of such technology is beyond the scope of our paper. In the following, we describe the main components that make up M-Aegis.

### 3.3.1 Layer 7.5 (L-7.5)

M-Aegis creates a novel and conceptual layer called Layer 7.5 (L-7.5) to interpose itself between the user and the TCA. This allows M-Aegis to implement a true end-to-end encryption (E2EE) without ever exposing the plaintext data to the TCA, which is considered untrusted, while maintaining the TCA's original functionalities and user experience, fulfilling the second design goal. L-7.5 is built by creating a transparent window that is always-on-top. This technique is advantageous in that it provides a natural way to handle user interaction, thus preserving user experience without the need to reverse-engineer the logic of TCAs or the network protocols used by the TCAs to communicate with their respective cloud service backends, fulfilling the third design goal.

There are basically three cases of user interactions to handle. The first case considers interactions that do not involve data confidentiality (e.g. deleting or relabeling email). Such input do not require extra processing/transformation and can be directly delivered to the underlying TCA. Mimic-GUIs for display such as that on the left in Fig. 2 are examples for this case. Such click-through behavior is a natural property of transparent windows, and helps M-Aegis maintain the fluency in user interaction.

The second case considers interactions that involve data confidentiality (e.g. entering message or searching encrypted email). Such input requires extra processing (e.g. encryption and encoding operations). For such cases, M-Aegis places opaque GUIs that "mimics" the GUIs on the TCA, which are purposely painted in different colors for two reasons: (a) as a placeholder for user input so that it does not leak to the TCA, and (b) for user's visual feedback. Mimic-GUIs for the subject and content as seen in Fig. 3 are examples for this case. Since L-7.5 is always on top, this provides the guarantee that user input always goes to the mimic-GUIs instead of those of the TCA's.

The third case considers interactions with control GUIs (e.g. clicking on send buttons). Such input requires user action to be "buffered" while the input in the second case is being processed before being relayed to the actual control GUI. For such cases, M-Aegis creates semi-transparent mimic-GUIs that register themselves to absorb/handle user clicks/taps. Again, these mimic GUIs are painted with a different color to provide visual cue to the user. Examples of these include the purple search button in the left figure in Fig. 2 and the purple send button in Fig. 3. Note that our concept of intercepting user input is similar to that of ACG's [60] in capturing user intent, but our application of user intent differs.

### 3.3.2 UIA Manager (UIAM)

To be fully functional, there are certain capabilities that M-Aegis requires but are not available to normal apps. First, although M-Aegis itself is confined within the OS' app sandbox, it must be able to figure out which TCA the user is currently interacting with. This is important so that the specific logic to handle the TCA can be properly invoked. Additionally, this is important to help M-Aegis clean up the screen when the TCA is terminated. Second, M-Aegis requires information about the GUI layout for the TCA it is currently handling. This allows M-Aegis to properly render mimic GUIs on L-7.5 to intercept user I/O. Third, although isolated from the TCA, M-Aegis

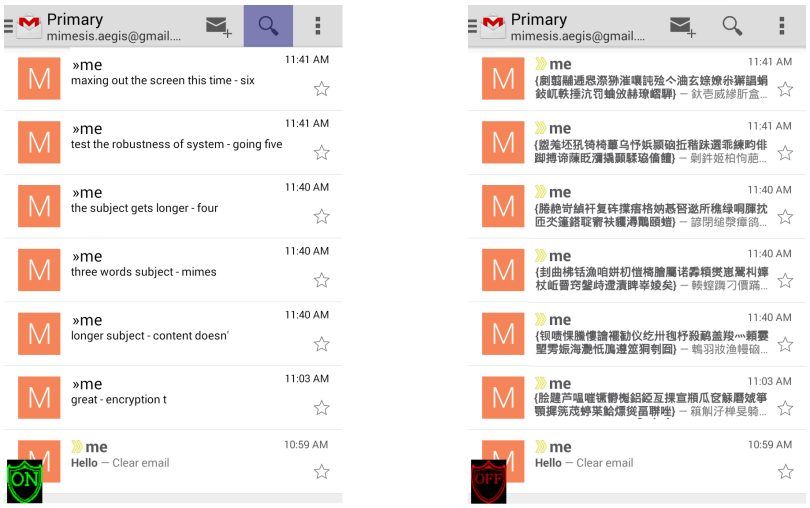


Figure 2: The figure on the left illustrates how user perceives the Gmail preview page when M-Aegis is turned on. The figure on the right illustrates the same scenario but with M-Aegis turned off. Note that the search button is painted with a different color on M-Aegis is turned on.

must be able to communicate with the TCA to maintain functionality and ensure user experience is not disrupted. For example, for user input, it must be able to relay user clicks to the TCA, and eventually send encrypted data to the TCA, and click on TCA's button on behalf of the user; for output on screen, it must be able to grab the ciphertext so that we can decrypt it and then render it on L-7.5.

M-Aegis extracts certain features from the underlying OS's accessibility framework, which are exposed to developers in the form of User Interface Accessibility/Automation (UIA) library. We are then not only able to know which TCA is currently executing, but we can also query the GUI tree of the TCA to get detailed information about how the page is being laid out (e.g. location, size, type, and resource-id of the GUIs). More importantly, we are able to get the information about the content of these GUI items.

Exploiting UIA is advantageous to our design as compared to other methods of grabbing information from the screen, e.g. OCR. Besides having perfect content accuracy, our technique is not limited by screen size. For example, even though the screen size may prevent full text to be displayed, the UIAM is still able to grab the text in its *entirety* through the UIA libraries, allowing us to comfortably apply decryption to the ciphertext.

We thus wrap around all these capabilities and advantages to build a crucial component of M-Aegis called the UIA manager (UIAM).

#### 3.3.3 Per-TCA Logic

M-Aegis can be extended to support many TCAs. For each TCA of interest, we build a per-TCA logic as an extension module. Basically, the per-TCA logic is responsible to render the specific mimic-GUIs according to information it queries from the UIAM. Therefore, the per-TCA logic is responsible for handling direct user input. Specifically, it decides whether the user input will be directly passed to the TCA or be encrypted and encoded before doing so. This ensures that the TCA never obtain the plaintext data while user interaction is still in plaintext mode. The per-TCA logic also intercepts button clicks so that it can then instruct UIAM to emulate the user's action on the button on the underlying TCA. Besides that, the per-TCA logic also decides which encryption and encoding scheme to use according to the type of TCA it is handling. For example, encryption and encoding schemes for handling email apps would differ from that of messenger apps.

#### 3.3.4 Cryptographic Module

M-Aegis' cryptographic module is responsible for providing encryption/decryption and cryptographic hash capabilities to support our searchable encryption scheme (described in detail later) to the per-TCA logic the so that M-Aegis can transform/obfuscate messages through E2EE operations. Besides standard cryptographic primitives, this module also includes a searchable encryption scheme to support search over encrypted email that works *without server modification*. Since the discussion of any encryption scheme is not complete without en-

cryption keys, key manager is also a part of this module.

**Key Manager.** M-Aegis has a key manager per TCA that manages key policies that can be specific to each TCA according to user preference. The key manager supports a range of schemes, including simple password-based key derivation functions to derive symmetric keys (that we assume to be shared out of band), which we currently implement as default, to more sophisticated PKI-based scheme for users who prefer stronger security guarantees and do not mind the additional key set-up and exchange overheads. However, the discussion about the best key management/distribution policy is beyond the scope of this paper.

**Searchable Encryption Scheme (EDESE).** There are numerous encryption schemes that support keyword search [49, 64, 48, 35, 37, 32, 55]. These schemes exhibit different tradeoffs between security, functionality and efficiency, but *all* of them require modifications on the server side. Schemes that make use of inverted index [37] are not suitable, as updates to inverted index cannot be practically deployed in our scenario.

Since we cannot assume server cooperation (consistent with our threat model in Sect. 3.2), we designed a new searchable encryption scheme called easily-deployable efficiently-searchable symmetric encryption scheme (EDESE). EDESE is an adoptation of a scheme proposed by Bellare et al. [27], with modifications similar to that of Goh's scheme [48] that is retrofittable to a non-modifying server scenario.

We incorporated EDESE for email applications with the following construct. The idea for the construction is simple: we encrypt the document with a standard encryption scheme and append MACs of unique keywords in the document. To prevent leaking the number of unique keywords we add as many "dummy" keywords as needed.

In order to achieve higher storage and search efficiency, we utilized a Bloom filter (BF) to represent the EDESE-index. Basically, a BF is a data structure that allows for efficient set-inclusion tests. However, such set-inclusion tests based on BFs are currently not supported by existing email providers, which only support string-based searches. Therefore, we devised a solution that encodes the positions of on-bits in a BF as Unicode strings.

Since the underlying data structure that is used to support EDESE is a BF, search operations are susceptible to false positives matches. However, this does not pose a real problem to users, because the false positive rate is extremely low and is completely adjustable. Our current implementation follows these parameters: the length of keyword (in bits) is estimated to be $k = 128$, the size of the BF array is $B = 2^{24}$, the maximum number of unique keywords used in any email thread is estimated to be

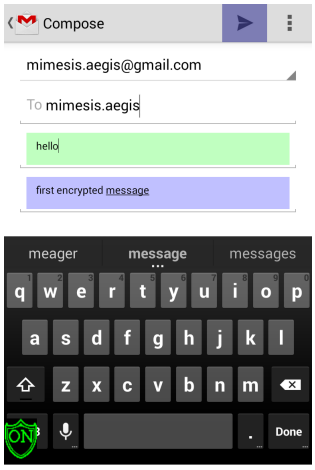


Figure 3: User still interacts with Gmail app to compose email, with M-Aegis' mimic GUIs painted with different colors on L-7.5.

$d = 10^6$, the number of bits set to 1 for one keyword is $r = 10$. Plugging in these values into the formula for false positive calculation [48], i.e. $(1 - e^{-rd/B})^r$, we cap the probability of a false positive $\delta$ to 0.0003.

We formally assess the security guarantees that our construction provides. In Appendix A, we propose a security definition for EDESE schemes and discuss why the existing notions are not suitable. Our definition considers an attacker who can obtain examples of encrypted documents of its choice and the results of queries of keywords of its choice. Given such an adversary, an EDESE scheme secure under our definition should hide all partial information about the messages except for the message length and the number of common keywords between any set of messages. Leaking the latter is unavoidable given that for the search function to be transparent to encryption, the output of a query has to be a part a ciphertext. But everything else, e.g., the number of unique keywords in a message, positions of the keywords, is hidden.

Given the security definition in Appendix A, we prove that our construction satisfies it under the standard notions of security for encryption and MACs in Appendix B. We also discuss the specific instantiations of encryption and MAC schemes that we use. The encoding scheme that we use is further described in Sect. 4.4.

## 3.4 User Workflow

To better illustrate how the different components in M-Aegis fits together, we describe an example workflow involved when a user composes and sends an email using stock Gmail app on Android with M-Aegis turned on:

1) When the user launches the Gmail app, the UIAM

notifies the correct per-TCA logic of the event. The per-TCA logic will then initialize itself to handle Gmail workflow.

2) As soon as Gmail is launched, the per-TCA logic will try to detect which state is Gmail app in (e.g. preview, reading, or composing email). This allows M-Aegis to properly create mirror-GUIs on L-7.5 to handle user interaction. For example, when a user is on the compose page, the per-TCA logic will mimic the GUIs of the subject and content fields (as seen in Fig. 3). The user then interacts directly with these mimic-GUIs in plaintext mode without extra effort. Thus, his workflow is not affected. Note that essential but subtle features like spell-check and autocorrect are still preserved, as they are innate features of mobile device's soft keyboard. Additionally, the "send" button is also mimicked to capture user intent.

3) As the user finishes composing his email, he clicks on the mimicked "send" button on L-7.5 once, as how he would normally do. Since L-7.5 receives the user input and not the underlying Gmail app, the per-TCA logic is able to capture this event and proceed to process the subject and the content.

4) The per-TCA logic selects the appropriate key to be used based on the recipient list and the predetermined key policy for Gmail. If a key cannot be found for this conversation, M-Aegis prompts the user (see Fig. 4) for a password to derive a new key. After obtaining the associated key for this conversation, M-Aegis will then encrypt these inputs and encode it back to text such that Gmail can consume it.

5) The per-TCA logic then requests the UIAM to fill in the corresponding GUIs on Gmail with the transformed text. After they are filled, the UIAM is asked to click the actual "send" button on behalf of the user. This provides a transparent experience to the user.

From this workflow, it should therefore be evident that from the user's perspective, the workflow of using Gmail remains the same, because of the mimicking properties of M-Aegis.

## 4 Implementation and Deployment

In this section, we discuss some important details of our prototype implementations. We implemented a prototype of M-Aegis using Java on Android, as an accessibility service. This is done by creating a class that extends the `AccessibilityService` class and requesting for `BIND_ACCESSIBILITY_SERVICE` permission in the manifest. This allows us to interface with the UIA library, building our UIAM. We discuss this in further detail in Sect. 4.2.

We then deployed our prototype on two Android phones from separate manufacturers, i.e. Samsung Galaxy Nexus, and LG Nexus 4, targeting several versions of Android, from Android 4.2.2 (API level 17) to the latest one: Android 4.4.2 (API level 19). The deployment is done on stock devices and OS, i.e. *without* modifying the OS or Android framework, or rooting, other than simple app installation. This demonstrates the ease of deployment and distribution of our solution. We have also implemented an M-Aegis prototype on Windows 7 to demonstrate interoperability and generality of approach, but we do not discuss the details here, as it is not the focus of this paper.

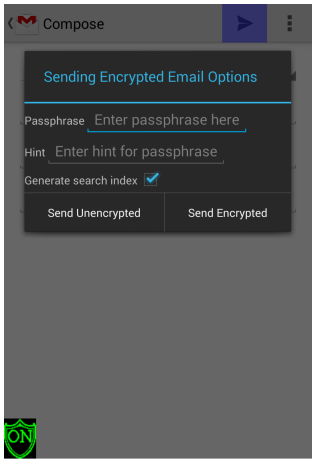


Figure 4: Password prompt when user sends encrypted mail for a new conversation.

As an interface to the user, we create a button that is always on top even if other apps are launched. This allows us to create a non-bypassable direct channel of communication with the user besides providing visual cue of whether M-Aegis is turned on or off.

For app support, we use Gmail as an example of email apps and WhatsApp as an example of messenger apps. We argue that it is easy to extend the support to other apps within these classes.

We first describe the cryptographic schemes that we deployed in our prototype, then we explain how we build our UIAM and create L-7.5 on Android, and finally discuss the per-TCA logic required to support both classes of apps.

### 4.1 Cryptographic Schemes

For all the encryption/decryption operations, we use AES-GCM-256. For password-based key generation algorithm, we utilized PBKDF2 with SHA-1 as keyed-hash message authentication code (HMAC). We also utilized HMAC-SHA-256 as our HMAC to generate tags for email messages (Sect. 4.4.1). These functionalities are available in Java's `javax.crypto` and

`java.security` packages.

In terms of key management, for the sake of usability, we implemented a password-based scheme as the default, and we assume one password for each group of message recipients. And we rely on the users to communicate the password to the receiving parties using out of band channel (e.g. in person or phone calls). For messaging apps, we implemented an authenticated Diffie-Hellman key exchange protocol to negotiate session keys for WhatsApp conversations. A PGP key is automatically generated for a user during installation based on the hashed phone number, and is deposited to publicly accessible repositories on the user's behalf (e.g. MIT PGP Key Server [10]). Further discussion about verifying the authenticity of public keys retrieved from such servers is omitted from this paper. Since all session and private keys are stored locally for user convenience, we make sure that they are never saved to disk in plaintext. They are additionally encrypted with a key derived from a master password that is provided by the user during installation.

## 4.2 UIAM

As mentioned earlier, UIAM is implemented based on UIA library. On Android, events that signify something new is being displayed on the screen can be detected by monitoring following events: `WINDOW_CONTENT_CHANGED`, `WINDOW_STATE_CHANGED`, and `VIEW_SCROLLED`. Upon receiving these events, the per-TCA logic is informed. The UIA library presents a data structure in the form of a tree with nodes representing UI components and the root being the top window. This allows the UIAM to locate all UI components on the screen.

Additionally, Android's UIA framework also provides us the ability to query for UI nodes by providing a resource ID in an intuitive way. For instance, the node that represents Gmail's search button can be found by querying for `com.google.android.gm:id/search`. More importantly, we do not need to guess the names of these resource IDs. Rather, we rely on a tool called UI Automater Viewer [17] (see how we use this tool in Sect. 4.4), which comes with the default Android SDK. Once the node of interest is found, all the other information about the GUI represented by the node can be found. This includes the exact location and size of text boxes and buttons on the screen.

Further, M-Aegis is able to programmatically interact with various GUIs of the TCA using the function `performAction()`. This is useful for it to click on the TCA's button on user's behalf after it has processed the user input.

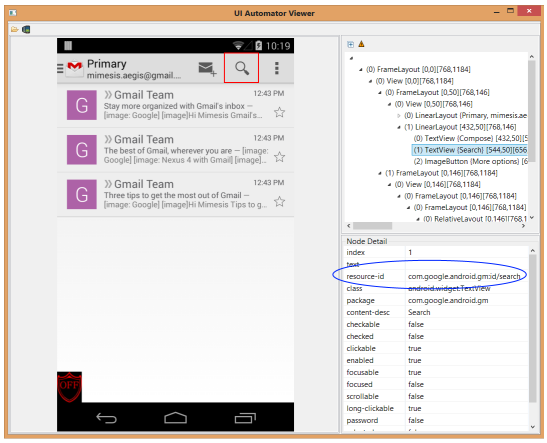


Figure 5: The UI Automator Viewer presents an easy to use interface for us to examine the UIA tree and determine the resource ID (blue ellipse) associated with the GUI of interest (red rectangle).

## 4.3 Layer 7.5

We implemented Layer 7.5 on Android as specific types of system windows, which are *always-on-top* of all other running apps. Generally, Android allows the creation of various types of system windows. In our case, we mainly focus on two, namely `TYPE_SYSTEM_OVERLAY` and `TYPE_SYSTEM_ERROR`; the first kind of window is for display-only and allows all tap/keyboard events to go to the underlying apps. In contrast, the user can interact with the second type of window. Finally, Android allows the use of any `View` objects for either type of window, and we use this to create our mimic-GUIs, of which we can set their size and location. We deliberately create our mimic-GUIs in different colors as a subtle visual cue to the users that they are interacting with M-Aegis, without distracting them from their original workflow.

## 4.4 Per-TCA Logic

From our experience of developing the per-TCA logic, the general procedure for development is as follow:

1) Understand what the app does. This allows us to identify which GUIs need to be mimicked for intercepting user I/O. For text-based TCAs, this is a very easy step because the core functionalities that M-Aegis needs to handle are limited and thus easy to identify, e.g. buffering user's typed texts and sending them to the intended recipient.

2) Using UI Automator Viewer [17], examine the UIA tree for the relevant GUIs of the TCA and identify signatures (in our case, it is the GUI's resource ID) for each TCA state. UI Automator Viewer allows us to inspect the UIA's tree through a graphical interface (as seen in

Fig. 5), which significantly cuts down on our development time. We basically rely on UI components that are unique to certain states (e.g. the "send" button signifies that we are in the compose state).

3) For each relevant GUI, we need to devise algorithms to extract either the location and content of ciphertext (for decryption and display), or the type, size and location of GUIs we need to mimic (e.g. the subject and content boxes in the Gmail compose UI). Again, this is done through UI Automator Viewer. For example, for the Gmail preview state, we query the UIA for nodes with ID `com.google.android.gm:/id/conversation_list` to identify all the UIA nodes corresponding to the preview item of each individual email, and from those we can extract all ciphertext on preview through the UIA).

4) Create event handlers for controls we mimic on L-7.5. For example, for the Gmail compose state, we need to listen for click/touch events to the "send" button we place on L-7.5 and carry out the process described in Sect. 3.3.3 to encrypt the email and send the ciphertext to the underlying TCA.

5) Identify ways that each relevant state can be updated. We found that updates can be handled with the following method: clear L-7.5 and extract all the information we need from that state and render everything again (this is how we handle scrolling). This is equivalent to redrawing all GUIs on L-7.5 based on the detected state.

There are two details that we need to pay attention to when developing per-TCA logic. First, we need to be careful about the type of input data that we feed to TCAs. Since most TCAs only accept input data in specific formats, e.g. text format, they do not support the input of random byte sequences as valid data. Therefore, the encrypted data must be encoded into text format before feeding it as input to the TCA. Conventionally, base64 encoding is used for such purposes. However, base64 encoding consumes too much on-screen real estate. To overcome this, we encoded the binary encrypted data into Chinese Japanese Korean (CJK) Unicode characters, which have a more efficient on-screen real estate consumption. To map the binary data into the CJK plane, we process the encrypted data at the byte granularity ($2^8$). For each byte, its value is added to the base of the CJK Unicode representation, i.e. 0x4E00. For example, byte 0x00 will be encoded as '一', and byte 0x01 will be represented as '丁'.

Second, M-Aegis can only function correctly if it can differentiate between ordinary messages and encrypted ones to decrypt message properly. We introduce markers into the encrypted data after encoding; in particular, we wrap each of the subject and content of a message using a pair of curly braces (i.e. $\{$, $\}$).

Next, we describe implementation details that are specific to these classes of apps. We begin by introducing the format of message we created for each class. Then we discuss other caveats (if any) that are involved in the implementation.

#### 4.4.1 Email Apps

We implemented support for Gmail on our prototype as a representative app of this category without loss of generality. For Gmail, we create two custom formats to communicate the necessary metadata to support M-Aegis' functionalities.
Subject: $\{Encode(ID_{Key}||IV||Encrypt(Subject))\}$
Content: $\{Encode(Encrypt(Content)||Tags)\}$

A particular challenge we faced in supporting decryption during the Gmail preview state is that only the beginning parts of both the title and the subject of each message are available to us. Also, the exact email addresses of the sender and recipients are not always available, as some are displayed as aliases, and some are hidden due to lack of space. The lack of such information makes it impossible to automatically decrypt the message even if the corresponding key actually exists on the system.

To solve these problems, when we encrypt a message, we include a key-ID ($ID_{Key}$) to the subject field (as seen in the format description above). Note that since the key-ID is not a secret, it need not be encrypted. This way, we will have all the information we need to correctly decrypt the little snippet displayed on the Gmail preview.

The *Tags* field is a collection of HMAC digests that are computed using the conversation key and keywords that exist in that particular email. It is then encoded and appended as part of the content that Gmail receives to facilitate encrypted search without requiring modification to Gmail servers.

#### 4.4.2 Messenger Apps

We implemented support for WhatsApp on our prototype as a representative app of this category without loss of generality. The format we created for this class of apps is simple, as seen below:
Message: $\{Encode(IV||Encrypt(Message))\}$

We did not experience additional challenges when supporting WhatsApp.

## 5 Evaluations

In this section, we report the results of experiments we performed to first determine the correctness of our prototype implementation, measure the overheads a user bears when using M-Aegis, and user acceptability of our approach.

## 5.1 Correctness of Implementation

Before we measure the performance, we manually verified that our app-support works correctly by navigating through different states of the app and check if M-Aegis creates L-7.5 correctly. Then, we also manually verified that the encryption and decryption operations on M-Aegis work correctly. We made sure that the plaintext is properly received at the recipient's end when the correct password is supplied. We also manually verified the correctness of our searchable encryption scheme by first planting specific keywords to be searched. Then, we performed search using M-Aegis and found no false negatives in the search result.

## 5.2 Performance on Android

The overhead that M-Aegis introduced to a user's workflow can be broken down into two main factors: i) the additional computational costs incurred during encryption and decryption of data, ii) the additional I/O operations when redrawing L-7.5. We measure the overhead by measuring the overall latency presented to the user in a few cases. We found that M-Aegis imposes negligible latency to the user.

All test cases were performed on one *stock* Android phone (LG Nexus 4), with the following specifications: Quad-core 1.5 GHz Snapdragon S4 Pro CPU, equipped with 2.0 GB RAM, running Android Kit Kat (4.4.2, with API level 19). Unless otherwise stated, each experiment is repeated for 10 times and the averaged result is reported.

For our evaluation, we only performed experiments on the setup for Gmail app because Gmail is representative of a more sophisticated TCA, therefore estimating the worst-case performance for M-Aegis. The other class, i.e. messenger apps, incurs much less overhead as the most involved operation is the encryption and decryption of short texts within a page.

### 5.2.1 Previewing Encrypted Email

There are additional costs involved in previewing encrypted emails on the main page of Gmail. The costs are broken down into times taken to i) traverse the UIA tree to identify preview nodes, ii) grab the ciphertext from the UIA node, iii) grab the associated key from key manager according to key ID, iv) decrypting the ciphertext, and v) rendering plaintext on L-7.5. We measure all these micro operations as an entirety instead by running a macro benchmarks.

For our experiment, we made sure that the whole preview page consists of encrypted emails (a total of 6 can fit the screen) to demonstrate the worst-case performance. Then, we measured the time taken to perform all the aforementioned micro operations. From the results, we found that for this case, on average, it takes an additional 76 ms to finally render plaintext on L-7.5. Note that this latency is well within the expected response time (50 - 150 ms), beyond which a user would notice the slowdown effect [63].

### 5.2.2 Composing and Sending Encrypted Email

First, we measured the extra time taken for a typical email to be encrypted and for our searchable encryption index to be built. We used the Enron Email Dataset [36] as a representation of how typical emails would be. Out of the dataset, we randomly picked 10 emails, with the following statistics: The average number of words in an email is 331, out of which 153 are unique. The shortest sampled email contains 36 words, out of which 35 are unique; while the longest sampled email contains 953 words, out of which 362 are unique.

In the worst case, i.e., with the longest sampled email, M-Aegis took around 205 ms in total to both encrypt and build search index. However, note that this is actually blended in with the network latency user will already perceive while sending out email.

### 5.2.3 Searching on Encrypted Emails

As a user usually inputs only about one to three keywords per search operation, the latency experienced when performing search is negligible. This is because the transformation of the actual keyword into indexes requires only the forward computation of one HMAC, which is almost instantaneous.

## 5.3 User Acceptability Study

This section describes the user study we performed to validate the hypothesis about user acceptability of our M-Aegis concept and prototype. Users are sampled from a population of college students, of which we consider as tech-savvy. Particularly, they must be able to proficiently operate smart phones and have had experience using Gmail app before. Each experiment was conducted with two identical smart phones, i.e. Nexus 4, both running Android 4.3, installed with stock Gmail app (v. 4.6). Only one of the devices has M-Aegis installed.

The set up of the experiment is as follows. First, we asked the user to perform a list of tasks: previewing, reading, composing, sending, and searching through email on a device that is not equipped with M-Aegis. Participants were asked to pay attention to the overall experience of performing such tasks using Gmail app and try their best to remember it. This served as the control experiment and to help participants recap the look and feel of the installed Gmail app.

Right after they are done, the participants were told to repeat the same set of tasks on another device which is equipped with M-Aegis. This was done with the intention that they were able to mentally compare the difference in user experience when interacting with the device equipped with M-Aegis.

After they were done with the second device, we ask the participants five questions, phrased in terms of do they find any difference in the preview page, reading, sending, and searching email, and if they felt that their overall experience using Gmail app on the second device is significantly different.

Upon finishing this, we debriefed the participants about the experiment process and explained the goal of M-Aegis. Lastly, we asked them whether they would use M-Aegis to protect the privacy of their data. The results we collected and report here are from 15 participants.

Altogether, we found that no participants noticed major difference between the two experiences using Gmail app. Only one participant noticed a minor difference in the email preview interface, i.e. L-7.5 did not catch up smoothly when scrolled. A different participant noticed a minor difference in the process of reading email, i.e. L-7.5 lags a little before covering up the ciphertext with mimic-GUIs. There were only two participants who found the process of sending email differs from the original one. When asked for details, they all indicated that the cursor when composing email was not working properly. After further investigation, we found that this is caused by a bug in Android's GUI framework rather than a fundamental flaw in our M-Aegis design.

However, despite the perceived minor differences when performing particular tasks, all participants indicated that they would use M-Aegis to protect the privacy of their data after understanding what M-Aegis is. This implies that they believe that the overall disturbance to the user experience is not large enough to impede adoption.

Since we recruited 15 users for this study, the accuracy/quality of our conclusion from this study lies between 80% and 95% (between 10 and 20 users) according to findings in [46]. We intend to continue our user study to further validate our acceptability hypothesis and to continuously improve our prototype based on the received feedback.

# 6 Discussions

## 6.1 Generality and Scalability

We believe that our M-Aegis architecture presents a general solution that protects users' data confidentiality, which is scalable in the following aspects:

**Across multiple cloud services.** In general, there are two main classes of apps that provide communication services, which cover a large number of apps that user care to protect, namely email and messenger apps. By covering apps in these two categories, we argue that we can already satisfy a large need of the user in protecting the privacy of their data.

All the different components of M-Aegis incur a one-time development cost. We argue that it is easy to scale across multiple cloud services, because the per-TCA logic that needs to be written is actually very little per new TCA that we want to support. This should be evident through the five general steps highlighted in Sect. 4.4. In addition, the logic we developed for the first TCA (Gmail) serves as a template/example to implement support other email apps.

**Across app updates.** Since the robustness of our UIAM construct (Sect. 4.2) gives us the ability to track all TCA's GUIs regardless of its state, we are able to survive app updates quite easily. In fact, our Gmail app support has survived two versions of updates without requiring major efforts to adapt.

As expected, resource ID names can change across updates. For example, when upgraded to Gmail app version 4.7.2, the resource ID name that identifies the sender's account name changed. But with a small fix to reidentify the resource ID using UI Automator Viewer, we quickly fixed it by changing the mapping in our per-TCA logic. Note that the logic in the per-TCA does not need to be modified. This is because the core functionality of the updated GUI would not change to something radically different. For example, a GUI associated with sender's account name would not suddenly has the functionality of a compose edit box.

## 6.2 Limitations

As mentioned earlier, M-Aegis is not designed to protect users against social engineering based attacks. Adversaries can try to trick users into entering sensitive information to the TCA while M-Aegis is turned off. Our solution is based on best effort by providing distinguishing visual cues to the user when M-Aegis is turned on and its L-7.5 is created. For example, when turned on, the mimic-GUIs that M-Aegis creates are in different color. Users can toggle M-Aegis' button on/off to see the difference (see Fig. 2). Note that M-Aegis' main button is always on top and cannot be drawn over by other apps. However, we do not claim that this fully mitigates the problem.

One of the constraints we faced while retrofitting a security solution to existing TCAs (not limited to mobile environment) is that data must usually be of the right format (e.g. strictly text, image, audio, or video). For exam-

ple, Gmail accepts only text (Unicode-compatible) as its subject, but Dropbox accepts any type of files, which can just be random blobs of bytes. Currently, other than the text format, we do not (yet) support other types of user data (e.g. image, audio and video). However, this is *not* a fundamental design limitation of our system. Rather, it is because of the unavailability of transformation functions (encryption and encoding schemes) that works for the aforementioned media types.

Additionally, unlike text, the transformation/obfuscation function that is applicable in our scenario that takes input of other type of data may also need to survive other process steps, such as compression. It is normal for TCAs to perform compression on other multimedia to save bandwidth and/or storage. For example, Facebook is known to compress/downsample the actual image that the user uploads despite choosing the HD option.

While the proper application of encryption schemes allows us to protect user's data privacy, the confidentiality guarantee that we provide excludes risks at end-point themselves. For example, poor random number generator can potentially weaken the cryptographic schemes that we applied. Further, it is currently unclear how our text transformations will affect the server's effectiveness in performing spam filtering.

Another limitation of our system is that it currently does not tolerate typographical error during search. However, we would like to point out that this is a very unlikely scenario, given that soft keyboards on mobile devices come with spell check and autocorrect features. Again, this is not a flaw with our architecture; rather, it is because of the unavailability of encryption schemes that tolerate typographical error search without requiring server modification.

## 7 Conclusions

In this paper we presented *Mimesis Aegis (M-Aegis)*, a new approach to protect private user data in public cloud services. M-Aegis provides strong isolation and preserves user experience through the creation of a novel conceptual layer called *Layer 7.5 (L-7.5)*, which acts as a proxy between the apps (Layer 7) and the user (Layer 8). This approach allows M-Aegis to implement a true end-to-end encryption of user data while achieving three goals: 1) plaintext data is never visible to a client app, any intermediary entities, or the cloud provider; 2) the original user experience with the client app is preserved completely, from workflow to GUI look-and-feel; and 3) the architecture and technique are general to a large number of apps and resilient to app updates. We implemented a prototype of M-Aegis on Android that can support a number of popular cloud services (e.g. Gmail, Google Hangout, Facebook, WhatsApp, and Viber). Our user study shows that our system truly preserves both the workflow and the GUI look-and-feel of the protected applications, and our performance shows that users experienced minimal overhead in utilizing M-Aegis on Android.

## Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable comments. We also thank the various members of our operations staff who provided proofreading of this paper. This material is based upon work supported in part by the National Science Foundation under Grants No. CNS-1017265, CNS-0831300, CNS-1149051, and CNS-1318511 by the Office of Naval Research under Grant No. N000140911042, by the Department of Homeland Security under contract No. N66001-12-C-0133, and by the United States Air Force under Contract No. FA8650-10-C-7025. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Office of Naval Research, the Department of Homeland Security, or the United States Air Force.

## References

[1] Accessibility. `http://developer.android.com/guide/topics/ui/accessibility/index.html`.

[2] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html.

[3] Cryptocat. https://crypto.cat.

[4] Engineering Security Solutions at Layer 8 and Above. https://blogs.rsa.com/engineering-security-solutions-at-layer-8-and-above/, December.

[5] Gibberbot for Android devices. https://securityinabox.org/en/Gibberbot_main.

[6] Google Accessibility. `https://www.google.com/accessibility/policy/`.

[7] Google Chrome Mobile FAQ. https://developers.google.com/chrome/mobile/docs/faq.

[8] International technology - Open Systems Interconnection - Basic Reference Model: The Basic Model. http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf.

[9] Microsoft Accessibility. `http://www.microsoft.com/enable/microsoft/section508.aspx`.

[10] MIT PGP Public Key Server. http://pgp.mit.edu/.

[11] New privacy fears as facebook begins selling personal access to companies to boost ailing profits. http://www.dailymail.co.uk/news/article-2212178/New-privacy-row-Facebook-begins-selling-access-users-boost-ailing-profits.html.

[12] Secure texts for Android. https://whispersystems.org.

[13] Sniffer tool displays other people's WhatsApp messages. http://www.h-online.com/security/news/item/Sniffer-tool-displays-other-people-s-WhatsApp-messages-1574382.html.

[14] Snowden: Leak of NSA spy programs "marks my end". http://www.cbsnews.com/8301-201_162-57588462/snowden-leak-of-nsa-spy-programs-marks-my-end/.

[15] Symantec desktop email encryption end-to-end email encryption software for laptops and desktops. http://www.symantec.com/desktop-email-encryption.

[16] Ten Mistakes That Can Ruin Customers' Mobile App Experience. http://www.itbusinessedge.com/slideshows/show.aspx?c=96038.

[17] UI Testing — Android Developers. `http://developer.android.com/tools/testing/testing_ui.html`.

[18] Whatsapp is broken, really broken. http://fileperms.org/whatsapp-is-broken-really-broken/.

[19] Layer 8 Linux Security: OPSEC for Linux Common Users, Developers and Systems Administrators. http://www.linuxgazette.net/164/kachold.html, July 2009.

[20] Accessibility. `https://www.apple.com/accessibility/resources/`, February 2014.

[21] 107TH CONGRESS. Uniting and strengthening america by providing appropriate tools required to intercept and obstruct terrorism (usa patriot act) act of 2001. *Public Law 107-56* (2001).

[22] ACQUISTI, A., AND GROSS, R. Imagined communities: Awareness, information sharing, and privacy on the facebook. In *Privacy enhancing technologies* (2006), Springer, pp. 36–58.

[23] AMANATIDIS, G., BOLDYREVA, A., AND O'NEILL, A. Provably-secure schemes for basic query support in outsourced databases. In *DBSec* (2007), S. Barker and G.-J. Ahn, Eds., vol. 4602 of *Lecture Notes in Computer Science*, Springer, pp. 14–30.

[24] BADEN, R., BENDER, A., SPRING, N., BHATTACHARJEE, B., AND STARIN, D. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM Computer Communication Review* (2009), vol. 39, ACM, pp. 135–146.

[25] BEATO, F., KOHLWEISS, M., AND WOUTERS, K. Scramble! your social network data. In *Privacy Enhancing Technologies* (2011), Springer, pp. 211–225.

[26] BELLARE, M., BOLDYREVA, A., AND MICALI, S. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT* (2000), B. Preneel, Ed., vol. 1807 of *Lecture Notes in Computer Science*, Springer, pp. 259–274.

[27] BELLARE, M., BOLDYREVA, A., AND O'NEILL, A. Deterministic and efficiently searchable encryption. In *CRYPTO* (2007), A. Menezes, Ed., vol. 4622 of *Lecture Notes in Computer Science*, Springer, pp. 535–552.

[28] BELLARE, M., KOHNO, T., AND NAMPREMPRE, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Trans. Inf. Syst. Secur. 7*, 2 (May 2004), 206–241.

[29] BELLARE, M., AND NAMPREMPRE, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT* (2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer, pp. 531–545.

[30] BERTHOME, P., FECHEROLLE, T., GUILLOTEAU, N., AND LALANDE, J.-F. Repackaging android applications for auditing access to private data. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on* (2012), IEEE, pp. 388–396.

[31] BÖHMER, M., HECHT, B., SCHÖNING, J., KRÜGER, A., AND BAUER, G. Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage. In *Proceedings of the 13th international conference on Human computer interaction with mobile devices and services* (2011), ACM, pp. 47–56.

[32] BONEH, D., CRESCENZO, G. D., OSTROVSKY, R., AND PERSIANO, G. Public key encryption with keyword search. In *EUROCRYPT* (2004), C. Cachin and J. Camenisch, Eds., vol. 3027 of *Lecture Notes in Computer Science*, Springer, pp. 506–522.

[33] BORDERS, K., VANDER WEELE, E., LAU, B., AND PRAKASH, A. Protecting confidential data on personal computers with storage capsules. *Ann Arbor 1001* (2009), 48109.

[34] BORISOV, N., GOLDBERG, I., AND BREWER, E. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society* (2004), ACM, pp. 77–84.

[35] CHANG, Y.-C., AND MITZENMACHER, M. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security*, J. Ioannidis, A. Keromytis, and M. Yung, Eds., vol. 3531 of *Lecture Notes in Computer Science*. Springer, 2005, pp. 442–455.

[36] COHEN, W. W. Enron email dataset. http://www.cs.cmu.edu/ enron, August 2009.

[37] CURTMOLA, R., GARAY, J. A., KAMARA, S., AND OSTROVSKY, R. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security* (2006), A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds., ACM, pp. 79–88.

[38] DELTCHEVA, R. Apple, AT&T data leak protection issues latest in cloud failures. http://www.messagingarchitects.com/resources/security-compliance-news/email-security/apple-att-data-leak-protection-issues-latest-in-cloud-failures19836720.html, June 2010.

[39] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. Tech. rep., DTIC Document, 2004.

[40] DONG, X., CHEN, Z., SIADATI, H., TOPLE, S., SAXENA, P., AND LIANG, Z. Protecting sensitive web content from client-side vulnerabilities with cryptons. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 1311–1324.

[41] ELKINS, M. Mime security with pretty good privacy (pgp).

[42] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI* (2010), vol. 10, pp. 1–6.

[43] FAHL, S., HARBACH, M., MUDERS, T., BAUMGÄRTNER, L., FREISLEBEN, B., AND SMITH, M. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 50–61.

[44] FAHL, S., HARBACH, M., MUDERS, T., AND SMITH, M. Trust-split: usable confidentiality for social network messaging. In *Proceedings of the 23rd ACM conference on Hypertext and social media* (2012), ACM, pp. 145–154.

[45] FARB, M., LIN, Y.-H., KIM, T. H.-J., MCCUNE, J., AND PERRIG, A. Safeslinger: easy-to-use and secure public-key exchange. In *Proceedings of the 19th annual international conference on Mobile computing & networking* (2013), ACM, pp. 417–428.

[46] FAULKNER, L. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers 35*, 3 (2003), 379–383.

[47] FELDMAN, A. J., BLANKSTEIN, A., FREEDMAN, M. J., AND FELTEN, E. W. Social networking with frientegrity: privacy and integrity with an untrusted provider. In *Proceedings of the 21st USENIX conference on Security symposium, Security* (2012), vol. 12.

[48] GOH, E.-J. Secure indexes. *IACR Cryptology ePrint Archive* (2003).

[49] GOLDREICH, O., AND OSTROVSKY, R. Software protection and simulation on oblivious rams. *J. ACM 43*, 3 (1996), 431–473.

[50] GUHA, S., TANG, K., AND FRANCIS, P. Noyb: Privacy in online social networks. In *Proceedings of the first workshop on Online social networks* (2008), ACM, pp. 49–54.

[51] HENRY, S. Largest hacking, data breach prosecution in U.S. history launches with five arrests. http://www.mercurynews.com/business/ci23730361/largest-hacking-data-breach-prosecution-u-s-history, July 2013.

[52] JANG, Y., CHUNG, S. P., PAYNE, B. D., AND LEE, W. Gyrus: A framework for user-intent monitoring of text-based networked applications. In *NDSS* (2014).

[53] JEON, J., MICINSKI, K. K., VAUGHAN, J. A., FOGEL, A., REDDY, N., FOSTER, J. S., AND MILLSTEIN, T. Dr. android and mr. hide: fine-grained permissions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices* (2012), ACM, pp. 3–14.

[54] JIANG, X. Gingermaster: First android malware utilizing a root exploit on android 2.3 (gingerbread). `http://www.csc.ncsu.edu/faculty/jiang/GingerMaster/`.

[55] KAMARA, S., PAPAMANTHOU, C., AND ROEDER, T. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 965–976.

[56] KING, S. T., TUCEK, J., COZZIE, A., GRIER, C., JIANG, W., AND ZHOU, Y. Designing and implementing malicious hardware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats* (Berkeley, CA, USA, 2008), LEET'08, USENIX Association, pp. 5:1–5:8.

[57] KONTAXIS, G., POLYCHRONAKIS, M., KEROMYTIS, A. D., AND MARKATOS, E. P. Privacy-preserving social plugins. In *Proceedings of the 21st USENIX conference on Security symposium* (2012), USENIX Association, pp. 30–30.

[58] LUCAS, M. M., AND BORISOV, N. Flybynight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society* (2008), ACM, pp. 1–8.

[59] PEEK, D., AND FLINN, J. Trapperkeeper: the case for using virtualization to add type awareness to file systems. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems* (2010), USENIX Association, pp. 8–8.

[60] ROESNER, F., KOHNO, T., MOSHCHUK, A., PARNO, B., WANG, H. J., AND COWAN, C. User-driven access control: Rethinking permission granting in modern operating systems. In *Security and Privacy (SP), 2012 IEEE Symposium on* (2012), IEEE, pp. 224–238.

[61] SHAH, K. Common Mobile App Design Mistakes to Take Care. http://www.enterprisecioforum.com/en/blogs/kaushalshah/common-mobile-app-design-mistakes-take-c.

[62] SHENG, S., BRODERICK, L., HYLAND, J., AND KORANDA, C. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security* (2006).

[63] SHNEIDERMAN, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, fourth ed. Addison-Wesley, 2005.

[64] SONG, D. X., WAGNER, D., AND PERRIG, A. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy* (2000), pp. 44–55.

[65] TSUNOO, Y., SAITO, T., SUZAKI, T., SHIGERI, M., AND MIYAUCHI, H. Cryptanalysis of des implemented on computers with cache. In *Cryptographic Hardware and Embedded Systems-CHES 2003*. Springer, 2003, pp. 62–76.

[66] US-CERT/NIST. Cve-2013-4787. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-4787`.

[67] WHITTAKER, S., MATTHEWS, T., CERRUTI, J., BADENES, H., AND TANG, J. Am i wasting my time organizing email?: a study of email refinding. In *PART 5———Proceedings of the 2011 annual conference on Human factors in computing systems* (2011), ACM, pp. 3449–3458.

[68] WHITTEN, A., AND TYGAR, J. D. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th USENIX Security Symposium* (1999), vol. 99, McGraw-Hill.

[69] WOJTCZUK, R., AND TERESHKIN, A. Attacking intel® bios. *Invisible Things Lab* (2010).

[70] WU, C., ZHOU, Y., PATEL, K., LIANG, Z., AND JIANG, X. Airbag: Boosting smartphone resistance to malware infection. In *NDSS* (2014).

[71] XU, R., SAÏDI, H., AND ANDERSON, R. Aurasium: Practical policy enforcement for android applications. In *Proceedings of the 21st USENIX conference on Security symposium* (2012), USENIX Association, pp. 27–27.

# A Appendices

# A Easily-Deployable ESE

In this section, we formally define our cryptographic scheme, which allows for search in encrypted emails.

## A.1 Preliminaries

NOTATION AND CONVENTIONS. We denote by $\Sigma^*$ the set of all binary strings of finite length. If $x, y$ are strings then $(x,y)$ denotes the concatenation of $x$ and $y$ from which $x$ and $y$ are uniquely decodable. For integers $k, l$, where $k < l$ $[k \dots l]$ denotes the set $\{k, k+1, \dots, l\}$ and $[k]$ denotes $\{1, \dots, k\}$. If $\mathsf{S}$ is a finite set, then $s \xleftarrow{\$} \mathsf{S}$ denotes that $s$ is selected uniformly at random from $\mathsf{S}$. $s \in_{\mathrm{D}} \mathsf{S}$ denotes that $s$ is selected in a deterministic way such that no element is selected more than once If $s, \mathsf{S}$ are strings, then $s \in \mathsf{S}$ denotes that $s$ is a substring of $\mathsf{S}$.

PROVABLE SECURITY APPROACH. In this work we apply the provable security approach. Unlike the unproductive and cyclic trial-and-error approach to security, this methodology allows us to have protocols, whose security is provably guaranteed, as long as the assumption about the underlying hard problem remains true for computationally bounded adversaries. This approach consists of the following components. (1) A formal definition of a protocol's syntax. (2) A formal definition of the security task in question that includes a precise description of adversarial capabilities and when is the adversary considered successful. (3) A reduction proof showing that the only way to break the protocol according to the definition is by breaking the underlying problem, believed to be hard.

SYMMETRIC ENCRYPTION AND ITS SECURITY. A *symmetric encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with associated *message space* $\mathsf{MsgSp}$, is defined by three algorithms: The randomized *key generation* algorithm $\mathcal{K}$ returns a secret key $sk$. The (possibly) randomized or stateful *encryption* algorithm $\mathcal{E}$ takes the input secret key $sk$, and a plaintext $m \in \mathsf{MsgSp}$, and returns a ciphertext. The deterministic *decryption* algorithm $\mathcal{D}$ takes the secret key $sk$, and a ciphertext $C$ to return the corresponding plaintext, or a special symbol $\perp$ indicating that the ciphertext was invalid. The consistency condition requires that $\mathcal{D}_{sk}(\mathcal{E}_{sk}(m)) = m$, for all $sk$ that can be output by $\mathcal{K}$, and all $m \in \mathsf{MsgSp}$.

We now recall the standard cryptographic security notions for encryption, *indistinguishability against chosen-plaintext attacks (IND-CPA)*. It formalizes the requirement that even though an adversary may know some partial information about the data, no additional information is leaked (besides the message length).

Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme. For an adversary $A$ and a bit $b$, define the experiment $\mathbf{Exp}^{\text{ind-cpa-b}}_{\mathcal{SE}}(A)$ as follows. First the key $sk$ is generated by $\mathcal{K}$. Let $\mathcal{LR}$ (left-or-right) be the "selector" that on input $m_0, m_1, b$ returns $m_b$. The adversary $A$ is given access to the *left-right encryption oracle* $\mathcal{E}_{sk}(\mathcal{LR}(\cdot,\cdot,b))$ that it can query on any pair of messages of equal length in $\mathsf{MsgSp}$. The adversary's goal is to output a bit $d$, as its guess of the challenge bit $b$, and the experiment returns $d$ as well. The *ind*-cpa-*advantage* of an adversary $A$, $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(A)$, is defined as the difference between the probabilities of $\mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE}}(A)$ returning 1 and $\mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE}}(A)$ returning 1.

The scheme $\mathcal{SE}$ is said to be *indistinguishable against chosen-plaintext attack* or *IND-CPA*, if for every adversary $A$ with reasonable resources its ind-cpa advantage is small[2].

MESSAGE AUTHENTICATION CODES (MACS). A *message authentication code (MAC)* $\mathcal{M} = (\mathcal{K}, \mathcal{T})$ with associated *message space* $\mathsf{MsgSp}$ is defined by two algorithms. The randomized *key generation* algorithm $\mathcal{K}$ returns a secret key $sk$. The deterministic[3] *tagging* algorithm $\mathcal{T}$ takes the input secret key $sk$, and a plaintext $m \in \mathsf{MsgSp}$ to return a tag for $m$. For a message-tag pair $(m, \sigma)$, we say $\sigma$ is a valid tag for $m$, if $\sigma = \sigma'$, where $\sigma' \leftarrow \mathcal{T}_{sk}(m)$.

We will use the following security definition (which implies the unforgeability against chosen message attack). Let $\mathcal{M} = (\mathcal{K}, \mathcal{T})$ be a MAC scheme. Let $R$ be the set of all functions with the same domain and range as $\mathcal{T}$. $\mathcal{M}$ is called *pseudorandom or PRF secure*, if any adversary $A$ with reasonable resources and access to an oracle that it can query on messages in $\mathsf{MsgSp}$, has small *prf-advantage* $\mathbf{Adv}^{\text{prf}}_{\mathcal{M}}(A)$ defined as

$$\Pr\left[ sk \xleftarrow{\$} \mathcal{K} \ : \ A^{\mathcal{T}_K(\cdot)} = 1 \right] - \Pr\left[ g \xleftarrow{\$} R \ : \ A^{g(\cdot)} = 1 \right] .$$

A weaker security notion, IND-DCPA, is defined in [28]. The notion asks the scheme to hide all but equality of the underlying messages. It is defined similar to IND-CPA but the adversary's queries are required to have the same equality pattern among the "left" and "right" vectors.

## A.2 Definition and Security

**Definition A.1 [EDESE]** An *easily-deployed efficiently- searchable symmetric encryption scheme (EDESE)* $\mathcal{ESE}$ is associated with *message space* $\mathsf{MsgSp}$ and *keyword space* $\mathsf{KwSp}$. We assume that each message $m \in \mathsf{MsgSp}$ can also be viewed as a set of keywords, and the particular meaning should be clear from the context. (E.g. $|m|$ denotes the length of $m$ in bits, $|m|_{\mathsf{kw}}$ denotes the number of keywords in $m$ and $m_0 \cap m_1$ denotes the set of common keywords in $m_0$ and $m_1$.) We assume that given $m \in \mathsf{MsgSp}$ one can efficiently extract all keywords in it. An EDESE scheme is defined by four algorithms:

- The randomized *key generation* algorithm $\mathcal{K}$ returns a secret key $sk$.
- The (possibly) randomized or stateful *encryption* algorithm $\mathcal{E}$ takes input the secret key $sk$, and a plaintext $m \in \mathsf{MsgSp}$, and returns a ciphertext $c$. To use

[2]Here, and further in the paper, we call the resources of an algorithm (or adversary) "reasonable", if it runs for some reasonable amount of time (e.g. up to 10 years, or does $2^{60}$ basic operations in some fixed model of computation), and does reasonable number of oracle queries of reasonable length. We call the value of an advantage "small", if it is very close to 0 (e.g. $2^{-20}$.) In general, " reasonable" parameters depend on a particular application. We do not use asymptotic notation because the symmetric key primitives such as blockciphers are of fixed length.

[3]A MAC does not have to be deterministic, but most practical schemes are, and in this paper we consider only deterministic MACs.

the fact that a ciphertext is searchable by a server, just like any other text, we will use the notation $s \in c$ to denote that a substring $s$ is part of $c$.

- The deterministic *decryption* algorithm $\mathscr{D}$ takes the secret key $sk$, and a ciphertext $c$ to return the plaintext.
- The deterministic *query* function $\mathscr{Q}$ takes input the secret key $sk$, and a keyword $w \in \mathsf{KwSp}$, and returns a string $s$.

We require that

- $\mathscr{D}_{sk}(\mathscr{E}_{sk}(m)) = m$, for all $sk$ that can be output by $\mathscr{K}$, and all $m \in \mathsf{MsgSp}$.
- For all $sk$ that can be output by $\mathscr{K}$, all $m \in \mathsf{MsgSp}$ and all $w \in \mathsf{KwSp}$, we always have that $\mathscr{Q}_{sk}(w) \in \mathscr{E}_{sk}(m)$, if $w \in m$, and the probability of $\mathscr{Q}_{sk}(w) \in \mathscr{E}_{sk}(m)$, if $w \notin m$, is at most $\delta$.

The latter condition ensures that the search on a keyword will always return all ciphertexts of messages containing the keyword, and that the probability of a false-positive is at most $\delta$.

EDESE SECURITY. We construct the following indistinguishability-based security definition, called PRIV-CPA, for analyzing the security of EDESE schemes. Intuitively, this notion is similar to the standard IND-CPA notion with the additional condition that left-right queries preserve the number of common equal components between any sets of underlying messages. This is because for functionality we allow a scheme to leak that different messages share keywords. We first provide the definition and follow with discussion.

**Definition A.2** Let $\mathscr{ESE} = (\mathscr{K}, \mathscr{E}, \mathscr{D}, \mathscr{Q})$ be an EDESE scheme with message and keyword spaces $\mathsf{MsgSp}, \mathsf{KwSp}$. For $b \in \Sigma$ and adversary $A$, we define the experiment $\mathbf{Exp}^{\text{priv-cpa-}b}_{\mathscr{ESE}}(A)$ similar to $\mathbf{Exp}^{\text{ind-cpa-b}}_{\mathscr{ESE}}(A)$ except that $A$ has the additional restriction: if $(m_0^1, m_1^1), \ldots, (m_0^q, m_1^q)$ are the queries $A$ makes to its LR encryption oracle $\mathscr{E}(sk, \mathscr{LR}(\cdot,\cdot,b))$, then

- for all $i, j \in [q]$, $|m_0^i \cap m_0^j|_{\mathsf{kw}} = |m_1^i \cap m_1^j|_{\mathsf{kw}}$.

For an adversary $A$, define its *PRIV-CPA advantage* against $\mathscr{ESE}$, $\mathbf{Adv}^{\text{priv-cpa}}_{\mathscr{ESE}}(A)$, as

$$\Pr\left[\mathbf{Exp}^{\text{priv-cpa-1}}_{\mathscr{ESE}}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{priv-cpa-0}}_{\mathscr{ESE}}(A) = 1\right].$$

We say that $\mathscr{ESE}$ is *private under chosen-plaintext attacks* (PRIV-CPA-secure) if the PRIV-CPA advantage of any adversary against $\mathscr{ESE}$ is small.

**Remark A.3** Note that the adversary is able to obtain examples of encrypted documents of its choice and queries of keywords of its choice, even though it is not given explicit encryption or query oracles. The adversary can still do the former by querying the left-right encryption oracle on pairs of equal messages and the latter by querying the same oracle on, say, a pair of equal messages containing just one keyword. Then by the searchability requirement the resulting ciphertext will contain the result of the query for this keyword.

**Remark A.4** We do not study chosen-ciphertext security here as it can be achieved using the encrypt-then-MAC method [29]. I.e., the key includes an additional MAC key, a new ciphertext includes the MAC of the previous-scheme ciphertext, and the new decryption algorithm verifies the MAC before decrypting and outputting the message.

**Remark A.5** Our security definition only considers a single key, when in reality the server would store and search messages encrypted under multiple keys. For the standard security definitions for encryption and MACs security in the single-user setting implies security in the multi-user setting [26]. But the situation with security of searchable encryption in the multi-user setting is not as obvious; e.g. [27] conjectures that for their primitive single-user security does not imply multi-user security. But the difference is due to the public-user setting. In the full version of the paper we will formally prove that our PRIV-CPA implies security in the multi-user setting.

**Remark A.6** Our security notion is similar to that of Goh [48], but ours is stronger in that we allow the challenge documents to contain different number of words and we allow the adversary to pick an arbitrary number of challenge document pairs adaptively. We do not know if the latter makes the definition strictly stronger, but unless one proves otherwise it is important to consider more powerful adversaries. Also, for our primitive the security notion requires the queries to hide partial information about keywords (besides equality). Goh's definition does not explicitly require that, even though such security may be implied for certain types of schemes.

The security definitions for privacy-preserving MACs (IND-DCPA) [28] and efficiently-searchable encryption [23] require that only equality of plaintexts is leaked. As are they are not directly suitable for our primitive as we must consider sets of messages and overlaps between them.

## A.3 A PRIV-CPA-secure EDESE scheme

OUR CONSTRUCTION. We now construct an EDESE scheme with message and keyword spaces $\mathsf{MsgSp}, \mathsf{KwSp}$

and prove it secure under the above definition. Let $\mathscr{SE} = (\mathscr{K}^{\mathscr{SE}}, \mathscr{E}^{\mathscr{SE}}, \mathscr{D}^{\mathscr{SE}})$ be a standard symmetric encryption scheme with message space $\mathsf{MsgSp}$, where message can also be viewed as sets of keywords. Let $\mathscr{M} = (\mathscr{K}, \mathscr{T})$ be a deterministic MAC with message space $\Sigma^*$. Let $k$ be an integer parameter. Let $\text{encode}_e, \text{encode}_q$ be functions $\Sigma^* \to \Sigma^*$.

We define $\mathscr{ESE} = (\mathscr{K}, \mathscr{E}, \mathscr{D}, \mathscr{Q})$ as follows.

- $\mathscr{K}$ runs $sk_{\mathscr{M}} \xleftarrow{\$} \mathscr{K}^{\mathscr{M}}$ and $sk_{\mathscr{SE}} \xleftarrow{\$} \mathscr{K}^{\mathscr{SE}}$, and returns $sk_{\mathscr{M}} \| sk_{\mathscr{SE}}$.
- $\mathscr{E}_{sk_{\mathscr{M}} \| sk_{\mathscr{SE}}}(m)$ runs:
  - Let $m = \{w_1, \ldots, w_\ell\}$.
  - Let $n$ be the maximum number of unique keywords a message of length $|m|$ can have.
  - If $\ell < n$, then for $i \in [\ell+1, \ldots, n]$ $w_i \xleftarrow{\$} \Sigma^k$
  - $t_i \leftarrow \mathscr{T}_{sk_{\mathscr{M}}}(w_i)$ for $i \in [n]$
  - $t \leftarrow \text{encode}_e(t_1, \ldots, t_n)$
  - Return $t \| \mathscr{E}^{\mathscr{SE}}_{sk_{\mathscr{SE}}}(m)$.
- $\mathscr{D}_{sk_{\mathscr{M}} \| sk_{\mathscr{SE}}}(c)$ parses $c$ as $\mathtt{tags} \| c'$ and returns $\mathscr{D}^{\mathscr{SE}}_{sk_{\mathscr{SE}}}(c')$.
- $\mathscr{Q}_{sk_{\mathscr{M}} \| sk_{\mathscr{SE}}}(w) = \text{encode}_q(\mathscr{T}_{sk_{\mathscr{M}}}(w))$.

The idea for the construction is simple: we encrypt the document with a standard encryption scheme and append MACs of unique keywords in the document. To prevent leaking the number of unique keywords we add as many "dummy" keywords as needed.

We now discuss how to instantiate the encode functions so that the scheme satisfies the correctness requirements.

CHOOSING encode FUNCTIONS. We could simply let the $\text{encode}_e$ function above to output its inputs in lexicographical order and $\text{encode}_q$ be the identity function. Note that in this case it is easy to see that the construction satisfies the requirements of a EDESE scheme with the false-positives bound $\delta$ corresponding to the probability of a queried message colliding with a random $k$-bit message or of MACs of two messages being the same, and this is small due to the standard unforgeability property of a MAC scheme.

Now we discuss how to instantiate encode functions so that ciphertexts get more compact. We introduce some extra parameters for the scheme: the size of the initially empty Bloom filter bit array $B$ and the number of bits $r$ being set to 1 for each keyword (these determine the rate of false positives as we discuss below). Let $p$ be the length of MACs output by the algorithm $\mathscr{T}$ of $\mathscr{M}$. We assume that $p \geq r \log B$. Let $f$ be an injective function from $[B]$ to $\Sigma^*$. Then

- $\text{encode}_e(t_1, \ldots, t_n)$:
  - for every $i \in [n]$
    - Let $h_1, \ldots, h_r$ be the numbers defined by 1st,$\ldots$, $r$th $\log B$-bits of $t_i$ resp.
    - Set $h_1, \ldots, h_r$'th bits of Bloom filter array to 1.
  - For all positions $p_1, \ldots, p_v$ of the Bloom filter which are 1, output $f(p_1), \ldots, f(p_v)$.
- $\text{encode}_q(t_i)$:
  - Let $h_1, \ldots, h_r$ be the numbers defined by 1st,$\ldots$, $r$th $\log B$-bits of $t_i$ resp.
  - Set $h_1, \ldots, h_r$'th bits of Bloom filter array to 1.
  - For all positions $p_1, \ldots, p_u$ of the Bloom filter which are 1, output $f(p_1), \ldots, f(p_u)$.

It is easy to see that the scheme is correct. When the same keyword is encoded during encryption and query, its deterministic MAC is mapped the same way to the same set of positions in Bloom filter. If $\mathscr{M}$ is a PRF, then setting the bits to 1 using our method is equivalent to using $r$ independent functions to random bits. After inserting $d \leq |\mathsf{KwSp}|$ distinct keywords using $r$ mappings into an array of size $B$ the probability of a false positive $\delta$ is $1 - (1 - (1/B))^{rd} r \approx (1 - e^{-rd/B})^r$ [48].

We note that our scheme is more more efficient than that of [48]. To map a keyword to $r$ bits [48] uses $r$ independent MAC computations. We, on the other hand, only use one MAC computation and use the pseudorandomness property to construct an equivalent mapping.

SECURITY ANALYSIS. We now state the security of our scheme.

**Theorem A.7** If $\mathscr{M}$ is PRF and $\mathscr{SE}$ is IND-CPA-secure, then $\mathscr{ESE}$ is PRIV-CPA-secure.

The proof is in Appendix B.

Discussion the concrete instantiation of our construction can be found in Section 4.4.

# B Security Proof

We now prove Theorem A.7. Let $\mathscr{SE} = (\mathscr{K}^{\mathscr{SE}}, \mathscr{E}^{\mathscr{SE}}, \mathscr{D}^{\mathscr{SE}})$ be a standard symmetric encryption scheme with message space $\mathsf{MsgSp}$, where message can also be viewed as sets of keywords. Let $\mathscr{M} = (\mathscr{K}, \mathscr{T})$ be a deterministic MAC with message space $\Sigma^*$. Let $k$ be an integer parameter. As we mentioned, for simplicity we assume that $\text{encode}_e$ function in the construction outputs its inputs in lexicographical order. Let $\mathscr{ESE}$ be as per Construction A.3.

For an adversary $A$ we consider a sequence of experiments, starting with $\mathbf{Exp}_0(A) = \mathbf{Exp}^{\text{priv-cpa-0}}_{\mathscr{ESE}}(A)$

and ending with $\mathbf{Exp}_3(A) = \mathbf{Exp}^{\text{priv-cpa-1}}_{\mathscr{E},\mathscr{I},\mathscr{E}}(A)$. Let the left-right queries made by $A$ be $(m_0^1, m_1^1)$, $(m_0^2, m_1^2), \ldots, (m_0^q, m_1^q)$, where for $b \in \Sigma$ and $i \in [q]$ $m_b^i = \{w_b^{i,1}, \ldots, w_b^{i,qbi}\}$.

Then in $\mathbf{Exp}_0(A)$ the adversary is given the MACs and encryptions of $\{w_0^{1,1}, \ldots, w_0^{1,n}\}, m_0^1, \ldots, \{w_0^{q,1}, \ldots, w_0^{q,n}\}, m_0^q$ (the MACs are of the keywords in sets and the encryptions of messages, and the MACs are output in the lexicographical order). And in $\mathbf{Exp}_3(A)$ the adversary is given the MACs and encryptions of $\{w_1^{1,1}, \ldots, w_1^{1,n}\}, m_1^1, \ldots, \{w_1^{q,1}, \ldots, w_1^{q,n}\}, m_1^q$. Now consider experiment $\mathbf{Exp}_1(A)$ which is equivalent to $\mathbf{Exp}_0(A)$, except that $A$ is given encryptions of "right" messages (and the MACs are still of the "left" keywords': $\{w_0^{1,1}, \ldots, w_0^{1,n}\}, m_1^1, \ldots, \{w_0^{q,1}, \ldots, w_0^{q,n}\}, m_1^q$.

Clearly, if the difference between probabilities of $A$ outputting 1 in $\mathbf{Exp}_0(A)$ and $\mathbf{Exp}_1(A)$ is not small, then we can construct adversary $B_1$ breaking IND-CPA security of $\mathscr{I}\mathscr{E}$ with equal $\mathbf{Adv}^{\text{ind-cpa}}_{\mathscr{I}\mathscr{E}}(B_1)$. $B_1$ simply uses its own challenge oracle to simulate the encryptions of documents output by $A$ (which must be of the same length in each query pair), and simulates the MACs of the keywords (which are the same in both experiments) using the key it creates.

Now we consider experiment $\mathbf{Exp}_2(A)$ which is equivalent to $\mathbf{Exp}_1(A)$, except that all keywords which are unique (i.e. are not shared by different documents) are replaced with the corresponding "right" unique keywords. Note that due to the queries restriction in Definition A.2 there are equal number of such unique keywords in the left and right queries (excluding the low probability of collision in dummy keywords). We claim that if the difference between probabilities of $A$ outputting 1 in $\mathbf{Exp}_1(A)$ and $\mathbf{Exp}_2(A)$ is not small, then we can easily construct adversary $B_2$ breaking PRF security of $\mathscr{M}$ with equal $\mathbf{Adv}^{\text{prf}}_{\mathscr{M}}(B_2)$. In fact, IND-DCPA security [28] would be sufficient. $B_2$ would use its own challenge oracle to simulate MACs of the keywords (while preserving the equality pattern in its queries) and simulate the encryptions of the documents using the key it creates. And of course we could go with the stronger PRF adversary that we need for the efficient encoding.

We finally claim that $\mathbf{Exp}_2(A)$ and $\mathbf{Exp}_3(A)$ are indistinguishable for $A$ or, again, we can easily construct adversary $B$ breaking PRF security of $\mathscr{M}$. This is because the only difference is in the MACs of keywords which are shared between several messages. But since the MAC is deterministic, the resulting tags are output in the lexicographical order, and because of the restriction on the adversary $A$'s queries, we can see that these keywords underlying $A$'s queries in $\mathbf{Exp}_2(A)$ and $\mathbf{Exp}_3(A)$ have the same equality pattern. I.e. if we assume that the keywords in each set are sorted according to the corresponding MAC lexicographical positions, then $w_0^{i,j} = w_0^{k,j}$ iff $w_1^{i,j} = w_1^{k,j}$ for $i \neq k$ both $\in [q]$ and $j \in [n]$. Then it is easy to construct IND-DCPA adversary who can use its own challenge oracle to simulate the MACs. $B_3$ can do so without violating its restriction on using the oracle, which is for left and right vectors of messages to have the same equality pattern. And of course we could construct a stronger PRF adversary.

Putting it all together we can see that if there exist an adversary $A$ who can output 1 in $\mathbf{Exp}_3(A)$ and $\mathbf{Exp}_0(A)$ so that the difference between the corresponding probabilities is not small, and hence, by definition, $\mathbf{Adv}^{\text{priv-cpa}}_{\mathscr{E}\mathscr{I}\mathscr{E}}(A)$ is not small, then we can construct other adversaries $B_1, B_2$ with comparable resources such that

$$\mathbf{Adv}^{\text{priv-cpa}}_{\mathscr{E}\mathscr{I}\mathscr{E}}(A) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathscr{I}\mathscr{E}}(B_1) + \mathbf{Adv}^{\text{prf}}_{\mathscr{M}}(B_2) + \mathbf{Adv}^{\text{prf}}_{\mathscr{M}}(B_3),$$

and the statement of the theorem follows.