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 confidentiality of any data in public clouds can be violated, and consequently, while 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 in 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 the user experience through the creation of a conceptual layer called *Layer 7.5 (L-7.5)*, which is interposed between the application (OSI Layer 7) and the user (Layer 8). This approach allows M-Aegis to implement 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 app 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 GUIs to both intercept plaintext user input before transforming the input and feeding it to the underlying app, and to reverse-transform 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 when adopting M-Aegis on Android: imperceptible encryption/decryption latency and a low and adjustable false positive rate when searching over encrypted data.

1 Introduction

A continuously increasing number of users now 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. While the user's connectivity to the Internet is improved with mobile platforms, the problem of preserving data privacy while interacting with PCS remains 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 everyday life, the default method 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; but such trust is unjustified, if not misplaced. Incidents that demonstrate breach of this trust are easy to come by: 1) PCS providers are bound by law to share their users' data with surveillance agencies [14], 2) it is the business model of the PCS providers to mine their users' data and share it with third parties [11, 22, 24, 40], 3) operator errors [34] can result in unintended data access, and 4) data servers can be compromised by attackers [47].

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 the threats mentioned above. 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 come in various combinations of look and feel and features, we observed that many of these apps provide 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 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.

Therefore, 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 [65, 59]. A good solution must present clear advantages to the entire mobile security ecosystem. In particular it must account for these factors: 1) the users' ease-of-use, hence acceptability and adoptability; 2) the developers' efforts to maintain support; and 3) the feasibility and deployability of solution on a mobile system. From this analysis, we formulate three design goals 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 plaintext data or an 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 [41] provide good isolation, 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 [68, 26] 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 many mobile browsers do not support extensions [7], and mobile device users do not favor using mobile browsers [27] to access PCS. Therefore, we rule out conventional browser-plugin/extension-based solutions.

2. For a solution to be adoptable, it must preserve the user experience. We argue 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 a mobile platform would not work, because it forces users to use a separate and custom app, and it is impossible to recreate the richness and unique user experience of all existing text routing apps offered by various PCS providers today. In the context of mobile devices, PCS 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, 58]. Ultimately, users will choose apps that feel the most comfortable. To reduce interference with a 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 effort 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 introduction of PCS that are becoming indispensable in a user's everyday life, a good solution should also be able to integrate security features into apps without requiring reverse engineering of 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 three 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. Note that utilizing accessibility features for unorthodox purposes have been proposed by prior work [56, 48] that achieves different goals. L-7.5 extracts the GUI information of an 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, 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 sup-

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

port 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 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 is optionally processed) before being passed to an 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 is 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 similar to that of manipulating the DOM tree of a web app using JavaScript. While DOM tree manipulation only works for browsers, 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 requires resource id names to remain the same, which, through empirical evidence, often holds true. Even if a resource id changes, minimal effort is required to rediscover resource id names and remap them to the logic in M-Aegis. From our experience, M-Aegis does not require developer attention across 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 spell checking and in-app navigation are also preserved. However, despite user perception, the original app *never* receives 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 their ability to search, as it is an important aspect of user productivity [64]. 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.

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,

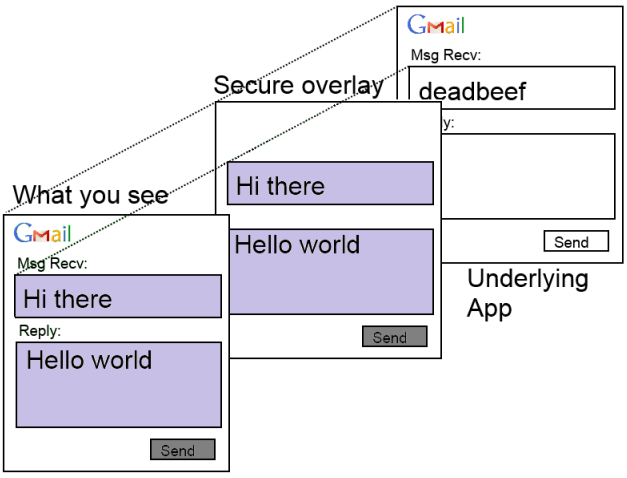

Figure 1: This diagram shows how M-Aegis uses L-7.5 to transparently reverse-transform the message "deadbeef" into "Hi there", and also allows a user to enter their plaintext message "Hello world" into M-Aegis's text box. To the user, the GUI looks exactly the same as the original app. When the user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

WhatsApp, and Facebook Chat. It protects data privacy by implementing E2EE that passes no plaintext to an app while also preserving the user experience and workflow. We also implemented a version of M-Aegis on the desktop to demonstrate the generality of our approach. Our initial performance evaluation and user study shows that users incur minimal overhead in adopting M-Aegis on Android. There is imperceptible encryption/decryption latency and a low and adjustable false positive rate when searching over encrypted data.

In summary, these are the major contributions of our 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 privacy when interacting with PCS by ensuring 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 to 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 in Sect. 1 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 design goals.

2.1 Standalone Solutions

There are many standalone solutions that aim to protect user data confidentiality. Solutions like PGP [15] (including S/MIME [37]), Gibberbot [5], TextSecure [12], SafeSlinger [41], and FlyByNight [55] provides secure messaging and/or file transfer through encryption of user data. These solutions provide good isolation from untrusted entities. However, since they are designed as standalone custom apps, they do not preserve the user experience, requiring users to adopt a new workflow on a custom app. More importantly, these solutions are not retrofittable to existing apps on the mobile platform.

Like M-Aegis, Cryptons [36] introduced a similarly strong notion of isolation through its custom abstractions. However, Cryptons assumes a completely different threat model that trusts PCS, and requires both server and client (app) modifications. Thus, Cryptons 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/Extension Solutions

Other solutions that focus on protecting user privacy include Cryptocat [3], Scramble! [24], TrustSplit [40], NOYB (None of Your Business) [46], and SafeButton [53]. Some of these assume different threat models, and achieve different goals. For example, NOYB protects a user's Facebook profile data while SafeButton tries to keep a 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 plugins/extensions, they are not applicable to the mobile platform.

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 server-side communication.

2.3 Repackaging/Rewriting Solutions

There is a category of work that repackages/rewrites an app's binary to introduce security features, such as Aurasium [68], Dr. Android [49], and others [26]. 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 they 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 platforms with cloud providers, we assume a very different threat model than those that focus on more robust permission model infrastructures and those that focus on controlling/tracking information flow, such as TaintDroid [38] and Airbag [67]. These solutions require changes to the underlying app, framework, or the OS, but M-Aegis does not.

Access Control Gadgets (ACG) [57] 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. Furthermore, ACG requires a modified kernel but M-Aegis does not.

Persona [23] 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 [43] and Gyrus [48] focus on different aspects of integrity protection of a user's data.

Tor [35] 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) [30] 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 our design goals. 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 the user experience by providing users 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) PCS providers can be compelled by law [21] to provide access to a user's sensitive data to law enforcement agencies [14]; 2) the business model of PCS providers creates strong incentive for them to share/sell user data with third parties [11, 22, 24, 40]; 3) PCS administrators who have access to the sensitive data may also compromise the data, either intentionally [14] or not [34]; and 4) vulnerabilities of the PCS can be exploited by attackers to exfiltrate sensitive data [47].
- Client-side apps. Since client-side apps are developed by PCS providers to allow a user to access their services, it follows that these apps are considered untrusted too.
- Middle boxes between a PCS and a client-side app. Sensitive data can also be compromised when it is transferred between a PCS and a client-side app. Incorrect protocol design/implementation may allow attackers to eavesdrop on plaintext data or perform Man-in-the-Middle attacks [39, 18, 13].

M-Aegis addresses the above threats by creating L-7.5, which it uses to provide end-to-end encryption (E2EE) for 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 the keystrokes of a user. We rely on the TCB to correctly handle I/O for M-Aegis, and to provide proper isolation between M-Aegis and 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 a user's data, but not its availability. Therefore, attacks that deny access to 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 malicious hardware [52], attacks against the hardware [66], the OS [50], the platform [63] 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 [48], Storage Capsules [29]) to minimize the attack surface against the TCB. Third, M-Aegis is designed to prevent any direct flow of information from an authorized user to untrusted entities. Hence, leakages through all side-channels [62] 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 the 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 components are beyond

the scope of our paper. Instead, M-Aegis deploys best-effort protection by coloring the UI components in L-7.5 differently from that of the default app 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 first. To achieve this, M-Aegis is designed to execute in a separate process, though it resides in the same OS as the target client app (TCA). Besides memory isolation, the filesystem of M-Aegis is also shielded from other apps by OS app sandbox protection.

Should a greater degree of isolation be desirable, an underlying virtual-machine-based system 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 platforms, and the adoption of such technology is beyond the scope of our paper. The main components that make up M-Aegis are as follows.

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 true end-to-end encryption (E2EE) without exposing plaintext data to the TCA 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 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. Such click-through behavior is a natural property of transparent windows, and helps M-Aegis maintain the look and feel of the TCA.

The second case considers interactions that involve data confidentiality (e.g. entering messages or searching encrypted email). Such input requires extra processing (e.g. encryption and encoding operations). For such cases, M-Aegis places opaque GUIs that "mimic" the GUIs over the TCA, which are purposely painted in different colors for two reasons: 1) as a placeholder for user input so that it does not leak to the TCA, and 2) for user visual feedback. Mimic GUIs for the subject and content as seen in Fig. 3 are examples of this case. Since L-7.5 is always on top, this provides the guarantee that user input always goes to a mimic GUI instead of the TCA.

The third case considers interactions with control GUIs (e.g. send buttons). Such input requires user action to be "buffered" while the input from the second case is being processed before being relayed to the actual control GUI of the TCA. 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 a visual cue to a 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 [57] 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 is confined within the OS' app sandbox, it must be able to determine with which TCA the user is currently interacting. This allows M-Aegis to invoke specific logic to handle the TCA, and helps 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 must be able to communicate with the TCA to maintain functionality and ensure user experience is not disrupted. For example, M-Aegis must be able to relay user clicks to the TCA, 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 capture ciphertext so that it 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. Using UIA, M-Aegis is not only able to know which TCA is currently executing, but it can also query the GUI tree of the TCA to get detailed information about how the page is laid out (e.g. location, size, type, and resource-id of the GUI components). More importantly, it is able to obtain information about the content of these GUI items.

Exploiting UIA is advantageous to our design as compared to other methods of information capture from the GUI, e.g. OCR. Besides having perfect content accuracy, our technique is not limited by screen size. For example,

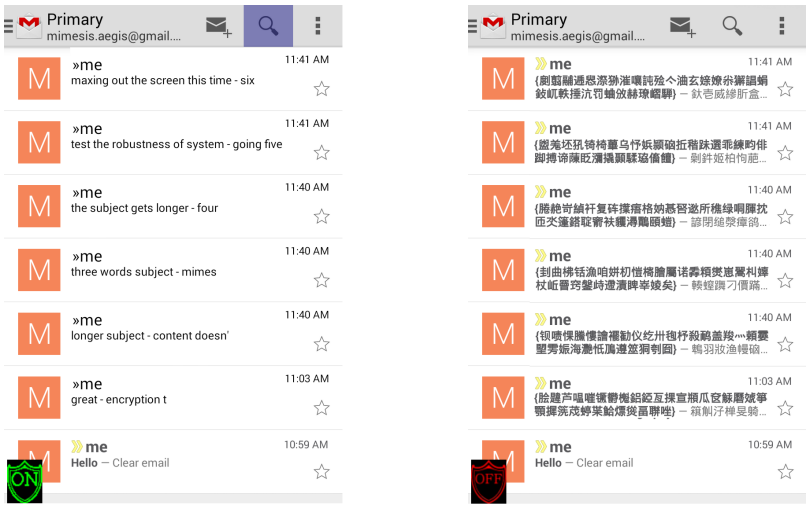

Figure 2: The figure on the left illustrates how a user perceives the Gmail preview page when M-Aegis is turned on. The figure on the right illustrates the same scenario but with M-Aegis turned off. Note that the search button is painted with a different color when M-Aegis is turned on.

even though the screen size may prevent full text to be displayed, M-Aegis is still able to capture text in its *entirety* through the UIA libraries, allowing us to comfortably apply decryption to ciphertext.

We thus utilize 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 per-TCA logic as an extension module. The per-TCA logic is responsible for rendering the specific mimic GUIs according to information it queries from the UIAM. Therefore, 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 obtains plaintext data while user interaction is in plaintext mode. Per-TCA logic also intercepts button clicks so that it can then instruct UIAM to emulate the user's action on the button in the underlying TCA. 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 encryption 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 (of which we assume the password to be shared out of band) to derive symmetric keys, 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 [45, 61, 44, 31, 33, 28, 51]. 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 [33] 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 adaptation of a scheme

proposed by Bellare et al. [25], with modifications similar to that of Goh's scheme [44] 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 HMACs of unique keywords in the document. We discuss the specific instantiations of encryption and HMAC schemes that we use in Sect. 4.1. To prevent leaking the number of unique keywords we add as many "dummy" keywords as needed. We present this construction in detail in the full version of our paper [54].

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 (refer to Sect. 4.4 for details).

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 $d = 10^6$, the number of bits set to 1 for one keyword is $r = 10$. Plugging in these values into the formula for false positive calculation [44], i.e. $(1 - e^{-rd/B})^r$, we cap the probability of a false positive δ to 0.0003.

We formally assess the security guarantees that our construction provides. In the full version of our paper [54], 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 our full paper [54], we prove that our construction satisfies it under the standard notions of security for encryption and HMACs.

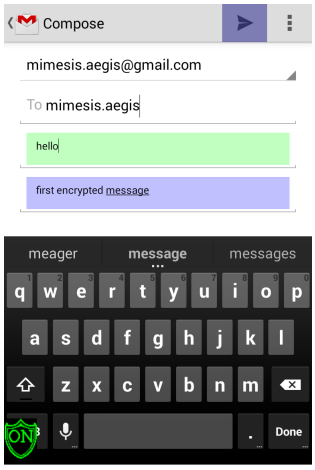

Figure 3: User still interacts with Gmail app to compose email, with M-Aegis' mimic GUIs painted with different colors on L-7.5.

3.4 User Workflow

To better illustrate how the different components in M-Aegis fit together, we describe an example workflow of a user composing and sending an email using the stock Gmail app on Android using M-Aegis:

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 the Gmail workflow.

2) As soon as Gmail is launched, the per-TCA logic will try to detect the state of Gmail app (e.g. preview, reading, or composing email). This allows M-Aegis to properly create mimic 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, the workflow is not affected. Note that essential but subtle features like spell check and autocorrect are still preserved, as they are innate features of the 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. 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 encryption 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 ob-

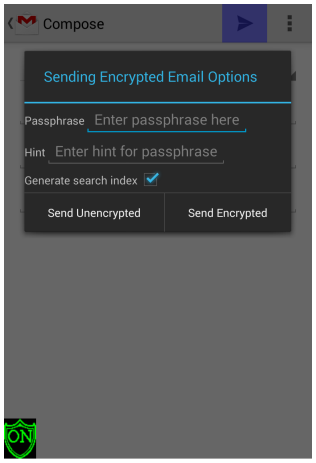

Figure 4: Password prompt when user sends encrypted mail for a new conversation.

taining 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 instructed 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 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 the `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 Android 4.4.2 (API level 19). The deployment was done on stock devices and OS, i.e. *without* modifying the OS, Android framework, or rooting. Only simple app installation was performed. 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.

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 an email app and WhatsApp as an example of a messenger app. 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 a password-based key generation algorithm, we utilized PBKDF2 with SHA-1 as the 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.

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. 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 using UIA libraries. On Android, events that signify something new being displayed on the screen can be detected by monitoring following the events: `WINDOW_CONTENT_CHANGED`, `WINDOW_STATE_CHANGED`,

and VIEW_SCROLLED. Upon receiving these events, per-TCA logic is informed. The UIA library presents a data structure in the form of a tree with nodes representing UI components with 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 the ability to query for UI nodes by providing a resource ID. For instance, the node that represents Gmail's search button can be found by querying for com.google.android.gm:id/search. More importantly, there is no need to guess the names of these resource IDs. Rather, a tool called UI Automator Viewer [17] (see 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.

M-Aegis is able to programmatically interact with various GUIs of a TCA using the function performAction(). This allows it t to click on a TCA's button on the user's behalf after it has processed the user input.

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. Android allows the creation of various types of system windows. We focus on two, TYPE_SYSTEM_OVERLAY and TYPE_SYSTEM_ERROR; the first is for display only and allows all tap/keyboard events to go to underlying apps. In contrast, the second type allows for user interaction. Android allows the use of any View objects for either type of window, and we use this to create our mimic GUIs, and 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 developing per-TCA logic, the general procedure for development is as follows:

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 trivial 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 a TCA and identify signatures (GUI resource IDs) for each TCA state. UI Automator Viewer allows inspection of the UIA tree through a graphical interface (as seen in Fig. 5), which reduces development time. We rely on UI components that are unique to certain states (e.g. the "send" button signifies that we are in the compose state).

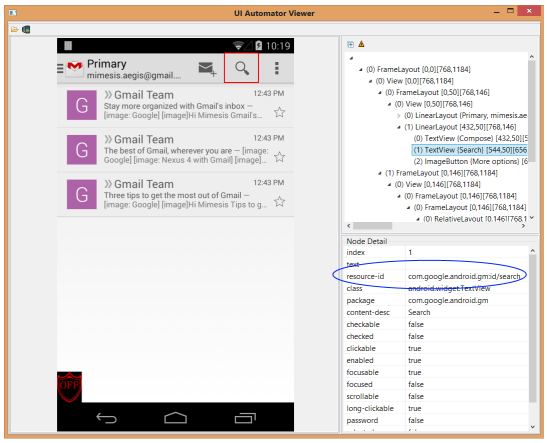

Figure 5: The UI Automator Viewer presents an easy to use interface to examine the UIA tree and determine the resource ID (blue ellipse) associated with a GUI of interest (red rectangle).

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 the preview window through the UIA).

4) Create event handlers for controls we mimic on L-7.5. For the Gmail compose state, we need to listen for click/touch events for the L-7.5 "send" button 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. Updates can be handled via the following method: clear L-7.5, extract all necessary information from the new state, and render again. This is equivalent to redrawing all GUIs on L-7.5 based on the detected state.

There are two details worth considering when developing per-TCA logic. First, careful consideration must be given about the type of input data fed to TCAs. Since most TCAs only accept input data in specific formats, e.g. text, they do not support the input of random byte sequences as valid data. Therefore, encrypted data must

be encoded into text format before feeding it as input to a TCA. Conventionally, base64 encoding is used for this purpose. 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 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 messages. We introduce markers into the encrypted data after encoding; in particular, we wrap 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. 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 encryption 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 subtext 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 a 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's servers.

4.4.2 Messenger Apps

We implemented support for WhatsApp on our prototype as a representative app of this category. 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 to determine the correctness of our prototype implementation, measure the overheads of M-Aegis, and user acceptability of our approach.

5.1 Correctness of Implementation

We manually verified M-Aegis's correctness by navigating through different states of the app and checking if M-Aegis creates L-7.5 correctly. We manually verified that the encryption and decryption operations of M-Aegis work correctly. We ensured that plaintext is properly received at the recipient's end when the correct password is supplied. We manually verified the correctness of our searchable encryption scheme by introducing specific search keywords. 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 factors: the additional computational costs incurred during encryption and decryption of data, and the additional I/O operations when redrawing L-7.5. We measure overhead by measuring the overall latency presented to the user in various use cases. We found that M-Aegis imposes negligible latency to the user.

All test cases were performed on a *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, API level 19). Unless otherwise stated, each experiment is repeated 10 times and the averaged result is reported.

For our evaluation, we only performed experiments for the setup of the Gmail app because Gmail is representative of a more sophisticated TCA, and thus indicates worst-case performance for M-Aegis. Messenger apps incur fewer overheads given their simpler TCA logic.

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 1) traverse the UIA tree to identify preview nodes, 2) capture ciphertext from the UIA node, 3) obtain the associated encryption key from the key manager, 4) decrypting ciphertext, and 5) rendering plaintext on L-7.5. We measure these operations as a single entity by running a macro benchmark.

For our experiment, we ensured that the preview page consists of encrypted emails (a total of 6 can fit on-screen) to demonstrate worst-case performance. We measured the time taken to perform all operations. We found, on average, it takes an additional 76 ms to render plaintext on L-7.5. Note that this latency is well within expected response time (50 - 150 ms), beyond which a user would notice the slowdown effect [60].

5.2.2 Composing and Sending Encrypted Email

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 [32] as a representation of typical emails. We randomly picked 10 emails. The average number of words in an email is 331, of which 153 are unique. The shortest sampled email contained 36 words, of which 35 are unique. The longest sampled email contains 953 words, of which 362 are unique.

With the longest sampled email, M-Aegis took 205 ms in total to both encrypt and build the search index. Note that this includes the network latency a user will perceive while sending an email, regardless of their use of M-Aegis.

5.2.3 Searching on Encrypted Emails

A user usually inputs 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 nearly instantaneous.

5.3 User Acceptability Study

This section describes the user study we performed to validate our hypothesis of user acceptability of M-Aegis. Users were sampled from a population of college students. They must be able to proficiently operate smart phones and have had previous experience using the Gmail app. Each experiment was conducted with two identical smart phones, i.e. Nexus 4, both running Android 4.3, installed with the stock Gmail app (v. 4.6). One of the devices had M-Aegis installed.

The set up of the experiment is as follows. 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 the Gmail app. This served as the control experiment.

Participants were then told to repeat the same set of tasks on another device that was 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 two devices.

We queried the participants if they found any difference in the preview page, reading, sending, and searching email, and if they felt that their overall experience using the Gmail app on the second device was significantly different.

We debriefed the participants about the experiment process and explained the goal of M-Aegis. 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.

We found that no participants noticed major differences between the two experiences using the Gmail app. 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 lag before covering ciphertext with mimic GUIs. There were only two participants that found the process of sending email differed from the original. When asked for details, they indicated that the cursor when composing email was not working properly. After further investigation, we determined this was a bug in Android's GUI framework rather than a fundamental flaw in M-Aegis's design.

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 [42]. We intend to continue our user study to further validate our acceptability hypothesis and to continuously improve our prototype based on received feedback.

6 Discussions

6.1 Generality and Scalability

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

Across Multiple Cloud Services. There are two main classes of apps that provide communication services, email and messenger apps. By providing functionality for apps in these two categories, we argue that M-Aegis can satisfy a large portion of user mobile security needs. 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 per-TCA logic that needs to be written is minimal per new TCA. 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 for other apps.

Across App Updates. Since the robustness of the UIAM construct (Sect. 4.2) gives M-Aegis the ability to track all TCA GUIs regardless of TCA state, M-Aegis is able to easily survive app updates. Our Gmail app support has survived two updates without requiring major efforts to adapt.

Resource ID names can change across updates. For example, when upgrading to Gmail app version 4.7.2, the resource ID name that identifies a sender's account name changed. Using UI Automator Viewer, we quickly discovered and modified the mapping in our TCA logic. Note that only the mapping was changed; the logic for the TCA does not need to be modified. This is because the core functionality of the updated GUI did not change (i.e., the GUI associated with a sender's account remained a text input box).

6.2 Limitations

As mentioned earlier, M-Aegis is not designed to protect users against social-engineering-based attacks. Adversaries can trick users into entering sensitive information to the TCA while M-Aegis is turned off. Our solution is best effort by providing distinguishing visual cues to the user when M-Aegis is turned on and its L-7.5 is active. For example the mimic GUIs that M-Aegis creates a different color. Users can toggle M-Aegis' button on or off to see the difference (see Fig. 2). Note that M-Aegis's 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 environments) is that data must usually be of the right format (e.g. strictly text, image, audio, or video). For example, Gmail accepts only text (Unicode-compatible) for an email subject, but Dropbox accepts any type of files, including random blobs of bytes. Currently, other than 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 these media types.

Unlike text, the transformation/obfuscation functions in M-Aegis for other type of data may also need to survive other process steps, such as compression. It is normal for TCAs to perform compression on multimedia to conserve bandwidth and/or storage. For example, Facebook is known to compress/downsample the image uploads.

The confidentiality guarantee that we provide excludes risks at the end points themselves. For example, a poor random number generator can potentially weaken the cryptographic schemes M-Aegis applies. It is currently unclear how our text transformations will affect a server's effectiveness in performing spam filtering.

Our system currently does not tolerate typographical error during search. However, we would like to point out that this is an unlikely scenario, given that soft keyboards on mobile devices utilize 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 an app (Layer 7) and a user (Layer 8). This approach allows M-Aegis to implement 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 preserves both the workflow and the GUI look-and-feel of the protected applications, and our performance evaluations show 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 vari-

ous 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] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

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

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

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

[46] 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.

[47] 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.

[48] 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).

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] LAU, B., CHUNG, S., SONG, C., JANG, Y., LEE, W., AND BOLDYREVA, A. Mimesis Aegis: A Mimicry Privacy Shield. http://hdl.handle.net/1853/52026.

[55] 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.

[56] 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.

[57] 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.

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

[59] 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).

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

[61] 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.

[62] 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.

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

[64] 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.

[65] 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.

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

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

[68] 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.