BabelCrypt: The Universal Encryption Layer for Mobile Messaging Applications

Ahmet Talha Ozcan[1], Can Gemicioglu[2], Kaan Onarlioglu[3], Michael Weissbacher[3], Collin Mulliner[3], William Robertson[3], and Engin Kirda[3]

[1] Middle East Technical University, Ankara
talha.ozcan@metu.edu.tr
[2] Sabanci University, Istanbul
cgemicioglu@sabanciuniv.edu,
[3] Northeastern University, Boston
{onarliog,mw,crm,wkr,ek}@ccs.neu.edu

Abstract. Internet-based mobile messaging applications have become a ubiquitous means of communication, and have quickly gained popularity over cellular short messages (SMS). Unfortunately, from a security point of view, free messaging services do not guarantee the privacy of users. For example, free messaging providers can record and store exchanged messages indefinitely to collect information about specific users. Moreover, these messages can be accessed by criminals who gain access to social media accounts. In this paper, we introduce BabelCrypt, a system that addresses the problem of automatically retrofitting arbitrary mobile chat applications with end-to-end encryption. Our system works by transparently interfacing with the original client applications supplied by the respective service providers. It does not require any modification to the individual applications, nor does it require any knowledge or customization for specific chat applications. BabelCrypt is able to automatically inject control messages in-band, using the underlying application's message exchange mechanism, and thus supports running arbitrarily complex encryption protocols such as OTR. We successfully used BabelCrypt with a number of popular messaging applications including Facebook Messenger, WhatsApp, and Skype. Our evaluation shows that BabelCrypt provides end-to-end security for arbitrary messaging applications while satisfactorily preserving the original user experience of the messaging application.

Keywords: Mobile messaging, Android security, privacy

1 Introduction

Internet-based mobile messaging applications that provide services such as the discovery of other users and exchanging text messages with them have become a ubiquitous means of communication. They have quickly gained popularity over cellular short messages (SMS) as such services are often free of charge, even when

roaming and switching to a different cellular network operator, and are available anywhere Internet connectivity is available.

Internet-based mobile messaging has also experienced huge growth in recent years due to the availability of inexpensive smartphones and tablets. The strong ties between text communication and smartphones become even more apparent when one considers that popular services (e.g., WhatsApp and Viber) only provide client software for smartphones and tablets. Today, many online social media services such as Facebook, Microsoft, Google, and Yahoo have followed the trend and are providing their own text-based communication service. Furthermore, a large number of video and voice communication services such as Skype and Viber are also providing chat-style text messaging features.

Unfortunately, there are also significant downsides of these free and always available communication services from a security point of view; in particular, user privacy suffers. While the underlying communication can easily be secured against eavesdropping by utilizing TLS, the service provider has full, unfettered access to every message exchanged through their infrastructure. Service providers can (ab)use this power to record and store exchanged messages indefinitely; for example, to collect information about specific users and serve them targeted ads [7]. Moreover, these messages can be accessed by rogue employees of the service provider, or criminals who gain access to social media accounts. Service providers can also be subpoenaed to hand over the stored data to government and law enforcement agencies that request access to a user's communication logs.

So far, there have been several attempts to secure Internet-based mobile communication. For instance, users of certain chat clients can install and use encryption plugins such as Off-the-Record (OTR) [9] to protect their privacy. However, these chat clients support only a limited set of communication protocols. In addition, many messaging services (e.g., Skype and Viber) use custom protocols that constantly evolve, forcing the user to update the chat client frequently, thus cutting out the development of third-party clients or plugins that support message encryption. Notably, recent research has proposed Mimesis Aegis [12], a system that addresses this problem by interposing a conceptual encryption layer between software and the users interacting with them. However, this approach requires development of specific logic for each chat client supported, and does not support automatic injection of messages into the communication channel, rendering it unable to support cryptographic protocols that involve, for instance, key exchange.

In this paper, we introduce BabelCrypt, a system that addresses the problem of retrofitting arbitrary mobile chat applications with end-to-end encryption. Our system works by transparently interfacing with the original client applications supplied by the respective service providers. It does not require any modification to the individual applications, nor does it require any knowledge or customization for specific chat applications. A significant advantage of BabelCrypt over comparable solutions is that it is also able to automatically inject control messages in-band, using the underlying application's message exchange mechanism, and thus supports running arbitrarily complex encryption protocols such as OTR.

BabelCrypt consists of two core components: an encrypting keyboard that transparently secures messages typed into the application by a user, and a decrypting display overlay that automatically analyzes the GUI of the chat application in real-time and adapts its appearance to mimic that of the underlying application. As a result, users interact with BabelCrypt in the exact same way they would with the original application. We show in Section 6 that BabelCrypt does not have a significant detrimental impact on the user experience and is easy to use, while providing transparent end-to-end encryption for the exchanged messages.

We implemented BabelCrypt for the popular Android platform. Our prototype implementation works on any Android device that runs Android version 4.x or later, and does not require any modification to the phone or superuser access to the operating system.

The main contributions of this paper are the following:

- We introduce BabelCrypt, a system for application-independent end-to-end encryption for Internet-based mobile text messaging. Our system protects messages against access by the messaging service providers.
- We show that BabelCrypt works by interfacing with the target chat application in the same way a user would, in a transparent manner, and does not significantly detract from the original user experience. BabelCrypt supports arbitrary chat applications, and does not require modification to or previous knowledge of individual applications. When using shared passwords for encryption, BabelCrypt does not require any setup procedure.
- We propose a technique for automatically injecting messages into the underlying chat application's message exchange system, enabling BabelCrypt to run cryptographic protocols such as OTR. In this mode, BabelCrypt only requires a simple one-time initialization routine per-installed application, only requiring the user to perform two clicks on the screen.
- We evaluate BabelCrypt using a wide range of text-based communication applications to demonstrate its generic applicability, performance, and usability.

2 Threat Model & Motivation

BabelCrypt aims to protect the confidentiality of communication between users of text-based online communication services in a transparent manner, without requiring drastic changes to the user experience or additional development effort. The threat model we consider for this work is divided into four distinct scenarios described below.

In the first scenario, we assume that the communication between the messaging application on the smartphone and the service provider can be intercepted and eavesdropped on by an attacker. This scenario covers possible mistakes in the usage of cryptographic primitives, which have been documented in previous

work [6]. For example, the application may fail to use transport layer encryption such as TLS for sensitive messages; otherwise, the application's use of cryptography might be implemented in an unsafe way that allows man-in-the-middle attacks.

The second scenario involves malicious communication service providers – that is, service providers that are benign, but that employ a business model based on monetizing information collected from their users. In our threat model, we assume that service providers have access to, and may record, all communication carried out through their infrastructure. They may access and use this information at any time, for example, to deliver targeted advertisements to their users. In addition, they may disclose the collected records to other entities, for instance, through company acquisitions or mergers, or to government and law enforcement agencies through subpoenas.

In the next scenario, we assume that user accounts may be compromised by an attacker. Chat services typically record conversations on the user's device or on their servers to provide conversation history and to implement a seamless handover between different devices owned by the same user. Therefore, an attacker can access entire conversation histories through compromised user credentials or stolen devices.

In the final scenario, we assume that third-party code embedded inside chat clients may freely intercept user communication. This might be due to malicious third-party code inclusion exploits, or implemented for benign purposes by the application developer – for example, to include advertisement libraries that scan for keywords and display targeted advertisements.

In all above scenarios, BabelCrypt aims to prevent inadvertent disclosure of users' communication records, keep their conversations confidential, and protect their privacy.

3 System Design

The design philosophy of BabelCrypt is to provide chat applications with end-to-end encryption in a completely transparent and generic manner, both from the perspective of the user and the underlying application. In particular, our system should satisfy the following design goals.

(D1) BabelCrypt must ensure that the user experience of interacting with the underlying chat application is not changed drastically.
(D2) BabelCrypt should be designed in a way that allows underlying chat application could remain oblivious to the presence of an encryption layer above it, or that it is transferring encrypted messages.
(D3) BabelCrypt must be independent of the specifics of the underlying chat application, and of the service provider infrastructure. This includes avoiding any form of modification to the underlying application's source code.

To this end, we designed BabelCrypt as a set of components that includes an extension to the system's software keyboard, and a GUI overlay over the

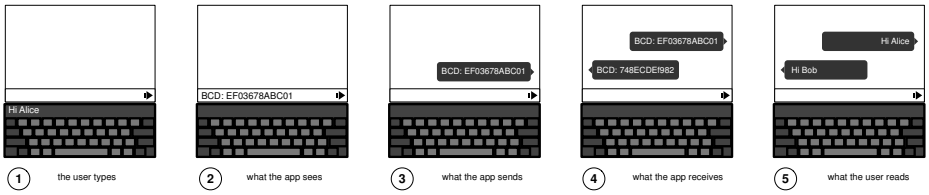

Fig. 1. The BabelCrypt system at work. The user types a message (1), when pressing send the keyboard encrypts the message and passes it to the application (2). The application sends the message (3). The application receives an encrypted message (4) and the overlay decrypts and displays the message to the user (5). Stages 3 and 4 are transparent to the user.

chat application screen. Users of chat applications type their messages through the BabelCrypt keyboard, just like they would interact with an ordinary keyboard, which encrypts the input on the fly and feeds it into the underlying application (D1). The GUI overlay automatically mimics the display of the chat application, and shows the decrypted plaintext where the underlying application would normally display the encrypted message (D1)(D2). During this process, BabelCrypt operates as an independent layer between the user and the target application, acting as a cryptographic conduit while remaining oblivious to both (D3).

Figure 1 provides an overview of BabelCrypt. In the rest of this section, we will describe the design of the core components of BabelCrypt in more detail.

3.1 BabelCrypt Keyboard

The primary interface between BabelCrypt and the user is the BabelCrypt keyboard. This component is an enhanced software keyboard that is a substitute for the operating system's default keyboard. It functions like a typical keyboard would, but also makes it possible to encrypt user input on-the-fly when a private conversation is requested. Using an additional mode switch button added at the bottom row of the keyboard, the user is able to turn the on-the-fly encryption on or off so that the same keyboard could be used system-wide as an ordinary keyboard with applications that do not necessitate encrypted input. The current mode of operation is indicated by a distinct visual cue – in particular, by changing the background color of the keyboard so that the user does not accidentally send unencrypted messages (see Figure 2).

When the encryption mode is on, instead of directly passing key presses to the underlying application, the BabelCrypt keyboard buffers all input, and displays it to the user in an auto-complete-bar like "plaintext field". Only when the user presses the "Return" or "Send" key is the entered text encrypted, and passed to the application. This ensures that the plaintext is never exposed to the underlying applications which may potentially leak them to the service provider without the user's knowledge. The aforementioned plaintext field makes it possible for the user to securely view and edit the text before it is sent, instead of typing blindly.

The keyboard is also tasked with encoding the ciphertext into printable characters, and splitting it into multiple messages of smaller chunks if necessary so that the underlying chat application can correctly transfer the encrypted message to the remote end.

3.2 BabelCrypt Display Overlay

The BabelCrypt display overlay is the component responsible for detecting text encrypted using BabelCrypt on the screen, and displaying it back to the user in plaintext.

This component has two main tasks. First, it continuously monitors the current foreground application window for changes to the GUI, which would indicate a new sent or received message being displayed. When such a change is triggered, this component accesses the underlying application's GUI tree, and traverses all visible nodes in it searching for encrypted text. Once ciphertext is found, BabelCrypt decodes it back to its original binary representation and decrypts it. BabelCrypt then automatically inspects the geometry of the GUI element that contains the ciphertext, overlays a textbox on top of it, and displays the decrypted text where it would originally have appeared in the chat application. In this way, we keep the original look and feel of the application, and do not change the user experience significantly.

BabelCrypt display overlay is able to perform these tasks thanks to the Android Accessibility Framework [10]. Using the accessibility API, BabelCrypt is able to access and inspect the GUI layout of the applications on the screen, without requiring modifications or the instrumentation of the application code.

Finally, similar to the keyboard component, BabelCrypt displays plaintext overlays in a distinct color to alert the user to the fact that the message has been sent encrypted (See Figure 2).

3.3 BabelCrypt Encryption Modes

BabelCrypt is designed with two encryption modes to support different use scenarios. Each mode provides different degrees of security guarantees and usability, allowing the users of the system the flexibility to pick the one that suits their needs. In this section, we describe these modes in more detail.

Encryption with Shared Secrets In this mode, BabelCrypt uses a basic shared secret scheme where the exchange of the cryptographic secret is delegated to the users (e.g., users share a password out of band).

The primary advantage of using this scheme is that no setup is necessary prior to running BabelCrypt; That is, users simply enter their passwords into a prompt, a key is derived from the password, and the users can immediately start exchanging messages. In addition, the communication history can be kept on the device or on the application servers in an encrypted form for future access by the

user. Finally, it is relatively easy to adapt this scheme to multi-user chat rooms by simply sharing the password with all involved parties.

Of course, shared secret encryption has the significant disadvantage of providing less strict security, including no forward secrecy nor authentication. Therefore, this encryption mode is suitable for users who would like to keep their chat histories, and would like a quick conversation without any setup process.

Encryption with Key-Agreement Protocols This second encryption mode allows users to run a cryptographic protocol over the target chat application's message exchange mechanism with the help of BabelCrypt. In this way, protocols of arbitrary complexity can be executed, for example, to perform an authenticated key exchange.

While the specific properties of such an encryption scheme depends on the actual protocol used, in general, this encryption mode makes it possible to hold a private conversation with stricter security guarantees such as authentication and perfect forward secrecy.

The primary disadvantage of this mode stems from the fact that cryptographic protocols typically require several steps of message exchanges before a session key for encryption can be established. However, performing such an exchange automatically would necessitate either establishing a separate out-of-band communication channel between two BabelCrypt endpoints, or using the in-band channel where text messages are also exchanged for this purpose. While the former is impractical, the latter is not directly possible since BabelCrypt does not have direct and automatic control over the communication channel; it can only input text into the underlying chat application through the user interacting with the keyboard.

As a result, in order to use this mode, the user needs to perform a simple one-time initialization step for every target application prior to using BabelCrypt with them. Specifically, the user needs to register with BabelCrypt the message entry box and the "Send" button of the application, so that BabelCrypt can subsequently inject protocol messages into the application and send them automatically without user interaction.

BabelCrypt handles both the task of registering these GUI components, and injecting messages, through the Android Accessibility Framework. Upon launching a chat application for the first time, the user needs to press a new "Set" button placed in the bottom row of the BabelCrypt keyboard which activates the registration mode. Next, the user touches the message entry box and the "Send" button on the screen, BabelCrypt intercepts these touch events, translates the touch coordinates to the corresponding GUI elements, and registers the resource identifiers corresponding to the message entry box and the button. Later, when message injection is required by the running protocol, BabelCrypt automatically traverses the application's GUI tree, locates the GUI elements corresponding to the saved identifiers, injects a message into the message entry box, and programmatically presses the "Send" button. Similarly, on the receiving side, BabelCrypt overlay traverses the GUI tree to find protocol messages displayed

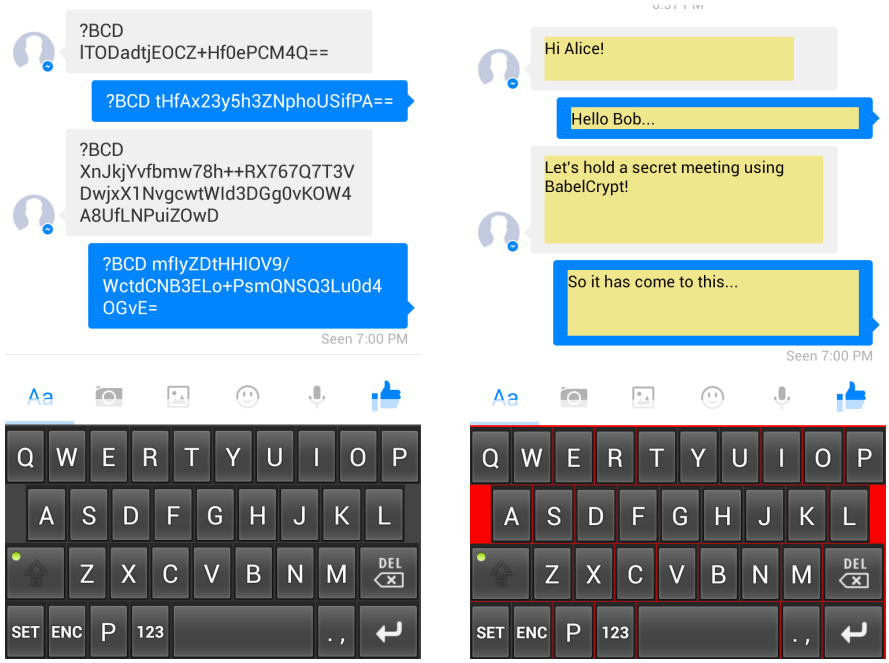

Fig. 2. An encrypted conversation displayed with BabelCrypt disabled on the left, and with BabelCrypt enabled on the right. Note that the background color of the keyboard and the overlay boxes change to indicate that a secure conversation is in progress.

by the chat application, and passes them to the encryption layer. In this way, after a simple initial setup, arbitrarily complex protocols can be automatically run without further user interaction.

4 Implementation

In the following, we provide details of our BabelCrypt prototype implementation and address some of the technical issues we elided in the previous sections.

4.1 Encryption Schemes

BabelCrypt currently supports one concrete encryption scheme for each of the two encryption modes it supports.

For encryption with shared secrets, we implemented a simple password-derived key scheme. Specifically, a 256-bit cryptographic key is derived from a pre-shared password using PBKDF2 with 10000 iterations. The encryption is performed using AES256 in CBC mode. IV values are transmitted along with the messages as we describe in the following sections.

For the more complex cryptographic protocol mode, we implemented the Off-the-Record Messaging (OTR) protocol, a protocol designed specifically for

text-based chat applications. It provides strong security guarantees such as perfect forward secrecy and deniable authentication, and is a good fit for BabelCrypt's security goals. Note that, however, OTR uses session keys that are periodically discarded, which makes it impossible to retrieve past conversation histories. OTR also does not support multi-user chat. As such, the simpler shared secret encryption mode still remains viable in different use cases.

4.2 Message Formats

BabelCrypt employs two different types of messages, data messages and control messages. Data messages carry encrypted user input, while the control messages are used for transmitting injected protocol messages.

Data messages could either be as a 3-tuple $\{BCD, IV, CIPHERTEXT\}$ if the shared-secret encryption mode, which requires sending the IV together with the message, is being used. Or, it could be a 2-tuple $\{BCD, OTRMSG\}$ if OTR is active. Here, BCD (BabelCrypt Data) is a special sequence that indicates that the payload of this message should be decrypted and displayed. When traversing the GUI tree, the BabelCrypt overlay component identifies encrypted user input by searching for this special tag. Examples of encrypted and decrypted messages are shown in Figure 2.

Similarly, control messages are formatted as a 3-tuple $\{BCC, ID, OTRMSG\}$, where BCC (BabelCrypt Control) is a different sequence tagging control messages. When the overlay component encounters such a message in the display, it passes the payload MSG to the encryption layer. Note that this exchange of control messages is visible in the chat application's display since control messages are transmitted through the chat application just like a normal conversation. Obviously, these messages are not human-readable, and hence, clutter the screen. Unfortunately, it is not possible for us to remove those messages from the screen as, for security reasons, the Android accessibility framework does not allow the modification of the GUI of the underlying applications. Therefore, in order not to confuse the user, BabelCrypt instead overlays a textbox on the displayed message, showing a notification informing the user that a cryptographic protocol is running and the contents of the message should be ignored.

A final issue arises from the fact that chat applications typically display both the incoming and outgoing messages on the screen. As a result, when a control message is injected by BabelCrypt into the application, it appears on the screens of both endpoints. However, control messages should only be *seen* and processed by the remote end. In order to prevent the sender from processing the control message destined for the other end, each control message also includes a randomly generated ID value. The sender inserts this to a set of IDs that should be ignored prior to sending the message and, as a result, the overlay component skips this message when searching for tagged entries on the screen. Similarly, once the control message is processed at the remote end, it is also inserted into an ignore list so that the same message is not processed multiple times, for example, when a user scrolls the screen and a previously processed message is displayed.

5 Limitations

By design, the underlying chat applications are completely oblivious of BabelCrypt. However, this can potentially lead to unexpected consequences when delivering encrypted messages. For instance, an application that does not allow the transfer of certain characters in the text, that transforms the messages in some way, or that otherwise has similar restrictions on the message format would break the integrity of BabelCrypt messages. Hence, the decryption on the remote end would be impossible. The keyboard component of our system is responsible for simple text encoding and splitting of messages, and we did not encounter applications requiring more sophisticated message handling in our tests; however, this possibility remains.

Another potential usability disadvantage is that some of the application features such as searching in the chat history would not be possible with BabelCrypt since the messages are stored in ciphertext. Likewise, features such as spell checking that could be performed inside an application need to be moved into the BabelCrypt keyboard as only the keyboard has access to plaintext input.

As previously noted, for secure communication protocols requiring automatic message injection, BabelCrypt necessitates a one-time setup during which the user interacts with the application's text entry box and message send button, and the system registers their GUI resource identifiers. While this approach makes BabelCrypt resonably robust against cosmetic changes to the underlying application's GUI, changes to resource identifiers may require the user to repeat the setup step, causing a minor disruption of the user experience.

Finally, BabelCrypt does not address the problem of sharing encrypted images, voice, or videos. This problem is outside the scope of this work.

6 Evaluation

In this section we describe our evaluation of BabelCrypt and show that it is compatible with prominent chat applications, that it does not incur a noticeable performance overhead, and that it does not have a significant negative impact on the user experience.

6.1 Applicability

In order to demonstrate that BabelCrypt works correctly with popular chat applications, we installed and extensively tested a set of popular applications found in the Android Marketplace. We verified that both shared password encryption and OTR modes correctly work in various applications such as the Facebook Messenger, WhatsApp, Tango, WeChat, Viber, and Skype. We note that although BabelCrypt is targeted at online messaging applications, it also works with SMS applications that display the messages as conversation flows. For instance, we verified that BabelCrypt works correctly with Go SMS [2].

Table 1. The results of a user study carried out with 40 participants, which demonstrate the usability of BabelCrypt.

Metric	Min	Q_1	Median	**Mean**	Q_3	Max	**Lower bound on 95% confidence interval**
Simplicity	75.00	75.00	100.00	**91.88**	100.00	100.0	**88.09**
Appearance	50.00	75.00	75.00	**75.62**	81.25	100.0	**70.04**
Likability	25.00	75.00	75.00	**74.38**	75.00	100.0	**70.14**

6.2 Performance

We were unable to reliably measure message round-trip times in our evaluation setup, due to factors such as network delays that lead to unpredictable latency in message delivery. Consequently, we opted to measure the performance by benchmarking the critical performance path of our system.

BabelCrypt has two execution paths that incur an overhead over the original chat application: the keyboard, and the display overlay. The keyboard is responsible for encrypting a single chunk of user input. However, the overlay component needs to traverse the entire GUI tree on each window content change, check GUI node contents for a match with the special BabelCrypt-tagged messages, and then process them, which typically includes decrypting several messages displayed on the screen at once. Thus, we chose to benchmark the overlay component since it represents the slowest path of execution in our system.

We have designed a macro benchmark that covers all of the above tasks performed by the BabelCrypt overlay. We triggered the whole process by manually sending encrypted messages to our test device from another remote device, and then measured the time for the overlay to finish detecting and processing all messages displayed on the screen. In our test setup, we used Facebook Messenger as the underlying chat application, and ran it on an off-the-shelf HTC One X smartphone. We have repeated the experiment 100 times and calculated the average runtimes. The results show that BabelCrypt incurred a performance overhead of **150.1 ms** on the average, with a standard deviation of **69.0 ms**, which indicates that the performance impact would not be detrimental to the user experience and that they would not have noticed a significant difference.

6.3 Usability

In order to evaluate the usability of the system and determine the impact of BabelCrypt on user experience, we conducted a user study with 40 participants. We confirmed that all of the participants are smartphone users, and that they are familiar with using at least one online messaging application.

We define our criteria for usability using three separate metrics. *Simplicity* is defined as the ease of interaction with the chat application, *appearance* is the perceived visual aesthetics of the application's user interface, and, finally,

likability captures the overall subjective experience of the user when interacting with the chat application.

In our study, we provided each participant with a Samsung Galaxy S3 smartphone loaded with Facebook Messenger, and asked them to exchange messages with a remote user. An experiment observer was tasked with responding to the participant's messages using another device, so that the participant can hold a realistic conversation with a human. The participants performed this task first with the vanilla messaging application, and then repeated the process with BabelCrypt enabled. They were then given an exit survey and asked to compare their experience with the messaging application in the two experiments. Specifically, they were asked three questions to compare the BabelCrypt-enabled system to the original application for each of our usability metrics defined above, and rate their experience on a 5-point Likert scale, where a higher score indicates a positive opinion (e.g., that BabelCrypt is as easy to use as the original application) and a lower score indicates a negative response (e.g., that BabelCrypt is very hard to use). After collecting user responses, we have normalized the points to a value between 0 and 100 to calculate a score for each usability metric. Finally, we computed the average scores, and analyzed the results to calculate the lower bound on a 95% confidence interval as to represent the worst-case scores. These results and five-number summaries of the collected data are presented in Table 1.

These results show that, BabelCrypt provides a degree of simplicity that is similar to the original messaging application. For the remaining two metrics, user feedback remains well above average, demonstrating that BabelCrypt does not have a significant negative impact on the user experience.

7 Related Work

The concept of confidential communication is not new, and solutions such as Pretty Good Privacy (PGP) [8] have been available for many years. PGP allows the encryption of arbitrary data, and it is most suitable for the encryption of email contents and attachments. Standalone systems such as PGP have good security properties, but they unfortunately suffer from poor usability. That is, users need to be familiar with the concept of public-key cryptography, and often need to install plugins that interface with the messaging application. Furthermore, if the application does not support a plugin interface, integration becomes difficult. To overcome these issues, other secure communication solutions have been developed. In the following, we discuss various systems that provide comparable solutions to BabelCrypt, and discuss the differences as well as the advantages and disadvantages.

There are several secure-messaging systems that were created specifically for smartphones, such as TextSecure [13], Threema [16], ChatSecure [15], and SilentCircle [14]. All of these services have been specifically designed to provide secure communication, but are *standalone* solutions. That is, users have to adjust to a new service and application (often with a new GUI), they have to install new software, and most importantly, they can only securely communicate with

contacts that are also using this service. In comparison, BabelCrypt has been designed to integrate with existing legacy services and the corresponding mobile applications. Therefore, the user can simply install BabelCrypt on her phone and continue to use existing applications such as WhatsApp and Skype without any disruption, or the need to add new contacts from scratch. However, our solution shares the above systems' limitation that all communicating parties need to install BabelCrypt on their devices.

Some chat clients, such as Pidgin [3] and Audium [1], come with a plugin architecture that allows third parties to develop application-specific plugins. Hence, security plugins such as Off-the-Record (OTR) [9] can be used to encrypt the communication between users even if the original protocol does not provide security features. Unfortunately, however, many popular messaging applications on smartphones (e.g., Viber, WhatsApp) do not provide a plugin architecture. BabelCrypt bridges this gap on the Android platform, and is able to secure arbitrary text-based messaging applications in a generic fashion. In other words, BabelCrypt can be seen as a universal plugin that is intended to work with any existing smartphone application.

In an alternative approach, repackaging-based systems such as Aurism [17], Dr. Android [11], I-ARM [5], and Bluebox [4] modify the original application binary in order to add privacy features such as message encryption. Repackaging solutions have the advantage that they run inside the application process, and thus, in theory, can completely integrate with the target application. Unfortunately, though, in practice, such solutions are often not very effective due to the high complexity of the messaging applications. Furthermore, the repackaging process has to be redone for every update of the target application, and these applications are sometimes be protected against reverse engineering attempts using obfuscation and other anti-reversing techniques. In comparison, BabelCrypt does not require any modification to the targeted messaging applications, and thus, works independently of the complexity of the underlying application. In addition, our solution is unlikely to be affected by application updates since we interact with the target application through the system keyboard, and by accessing the application GUI through the use of standard Android platform features.

A recent, and one of the conceptually closest systems to BabelCrypt is Mimesis Aegis [12]. Mimesis Aegis also aims to provide a solution that can work with arbitrary messaging applications on smartphones. The approach provides services such as message encryption and decryption and can provide a secure communication environment. However, it has the shortcoming that application-specific code needs to be developed for each application the user wishes to use (e.g., WhatsApp is not supported in the prototype as the authors have not implemented the application-specific GUI code). In contrast, BabelCrypt aims to be more generic, and works out-of-the-box with any arbitrary text-messaging application on a smartphone without the need to develop application-specific code. Moreover, a notable advantage of BabelCrypt over comparable solutions is that it is also able to automatically inject control messages in-band, using

the underlying application's message exchange mechanism, and thus, supports running arbitrarily complex encryption protocols such as OTR.

Recently, TextSecure and WhatsApp announced a collaboration to provide end-to-end security for messages exchanged using the WhatsApp mobile application and messaging service. While we laud this as a positive development for secure online communications, we also note that – to our knowledge – there are no plans to make WhatsApp's implementation of the TextSecure protocol open source or otherwise available for third party auditing. Therefore, BabelCrypt can provide an additional layer of assurance for privacy-concious users in this or similar scenarios.

8 Conclusion

Internet-based mobile messaging applications have become a ubiquitous and highly popular means of communication on mobile devices. Such services are often free-of-charge, and are available anywhere Internet connectivity is possible. Moreover, Internet-based mobile messaging has shown a significant growth in recent years due to the availability of inexpensive smartphones and tablets. Unfortunately, these messaging applications come at a cost in terms of privacy: Although the transport of the messages can be secured by the use of protocols such as TLS and are generally protected against man-in-the-middle attacks, the service provider can still (ab)use its power to record and store the exchanged messages indefinitely (e.g., to serve targeted ads to their users).

In this paper, we presented BabelCrypt, a generic, automated privacy-enhancing system that addresses the problem of retrofitting arbitrary mobile chat applications with end-to-end encryption. Our system works by transparently interfacing with the original client applications supplied by the respective service providers. BabelCrypt does not require modifications to the individual applications, nor does it require knowledge of or customization for specific chat applications. Compared to similar, existing systems, BabelCrypt has the advantage that it is able to automatically inject control messages in-band, using the underlying applications message exchange mechanism. Thus, it supports running arbitrarily complex encryption protocols such as OTR for applications that have not been designed with an open API (e.g., WhatsApp). Furthermore, BabelCrypt does not significantly alter the original user experience of the messaging applications, and thus provides a valuable and practical generic next step towards usable end-to-end security for mobile communications.

Acknowledgment This work was supported by the Office of Naval Research (ONR) under grant N000141210165, National Science Foundation (NSF) under grant CNS-1116777, and Secure Business Austria. The authors would like to thank Erik-Oliver Blass for insightful discussions and valuable feedback.

References

1. Audium. https://www.audium.im
2. Go SMS. http://gosms.goforandroid.com/
3. Pidgin, the universal chat client. https://www.pidgin.im
4. Bluebox Security: Bluebox. https://www.bluebox.com
5. Davis, B., Sanders, B., Khodaverdian, A., Chen, H.: I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications. In: IEEE Workshop on Mobile Security Technologies (2012)
6. Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C.: An Empirical Study of Cryptographic Misuse in Android Applications. In: ACM Conference on Computer and Communications Security (2013)
7. Feloni, R.: Facebook Sued For Allegedly Using Your Private Messages To Trigger Ads. http://www.businessinsider.com/facebook-sued-for-allegedly-using-your-private-messages-to-trigger-ads-2014-1 (January 2014)
8. Garfinkel, S.: PGP: Pretty good Privacy. O'Reilly Media, Inc (1995)
9. Goldberg, I.: Off-the-Record Messaging (OTR). https://otr.cypherpunks.ca/
10. Google: Accessibility — Android Developers. https://developer.android.com/guide/topics/ui/accessibility/
11. Jeon, J., Micinski, K.K., Vaughan, J.A., Fogel, A., Reddy, N., Foster, J.S., Millstein, T.: Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In: ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (2012)
12. Lau, B., Chung, S., Song, C., Jang, Y., Lee, W., Boldyreva, A.: Mimesis Aegis: A Mimicry Privacy Shield. In: USENIX Security Symposium (2014)
13. Open Whisper Systems: TextSecure. https://whispersystems.org
14. Silent Circle: Silent Text. https://www.silentcircle.com
15. The Guardian Project: ChatSecure. https://guardianproject.info/apps/chatsecure
16. Threema GmbH: Threema. https://www.threema.ch
17. Xu, R., Saïdi, H., Anderson, R.: Aurasium: Practical Policy Enforcement for Android Applications. In: USENIX Security Symposium (2012)