Cashtags: Protecting the Input and Display of Sensitive Data

Michael Mitchell and An-I Andy Wang, ***Florida State University;***
Peter Reiher, ***University of California, Los Angeles***

https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/mitchell

This paper is included in the Proceedings of the 24th USENIX Security Symposium

August 12–14, 2015 • Washington, D.C.

ISBN 978-1-939133-11-3

Open access to the Proceedings of the 24th USENIX Security Symposium is sponsored by USENIX

Cashtags: Protecting the Input and Display of Sensitive Data

Michael Mitchell, An-I Andy Wang
Florida State University

Peter Reiher
University of California, Los Angeles

Abstract

Mobile computing is the new norm. As people feel increasingly comfortable computing in public places such as coffee shops and transportation hubs, the threat of exposing sensitive information increases. While solutions exist to guard the communication channels used by mobile devices, the visual channel remains largely open. Shoulder surfing is becoming a viable threat in a world where users are often surrounded by high-power cameras, and sensitive information can be extracted from images using only modest computing power.

In response, we present Cashtags: a system to defend against attacks on mobile devices based on visual observations. The system allows users to safely access pieces of sensitive information in public by intercepting and replacing sensitive data elements with non-sensitive data elements before they are displayed on the screen. In addition, the system provides a means of computing with sensitive data in a non-observable way, while maintaining full functionality and legacy compatibility across applications.

1. Introduction

Shoulder surfing has become a concern in the context of mobile computing. As mobile devices become increasingly capable, people are able to access a much richer set of applications in public places such as coffee shops and public transportation hubs. Inadvertently, users risk exposing sensitive information to bystanders through the screen display. Such information exposure can increase the risk of personal, fiscal, and criminal identity theft. Exposing trade or governmental secrets can lead to business losses, government espionage, and other forms of cyber terrorism [12, 13, 14].

This problem is exacerbated by the ubiquity of surveillance and high-power cameras on mobile devices such as smartphones and on emerging wearable computing devices such as Google Glass [57]. Additionally, the trend toward multicore machines, GPUs, and cloud computing makes computing cycles more accessible and affordable for criminals or even seasoned hobbyists to extract sensitive information via off-the-shelf visual analysis tools [58].

This paper presents the design, implementation, and evaluation of Cashtags, a system that defends against shoulder surfing threats. With Cashtags, sensitive information will be masked with user-defined aliases, and a user can use these aliases to compute in public. Our system is compatible with legacy features such as auto correct, and our deployment model requires no changes to applications and the underlying firmware, with a performance overhead of less than 3%.

1.1 The shoulder-surfing threat

The threat of exposing sensitive information on screen to bystanders is real. In a recent study of IT professionals, 85% of those surveyed admitted seeing unauthorized sensitive on-screen data, and 82% admitted that their own sensitive on-screen data could be viewed by unauthorized personnel at times [1]. These results are consistent with other surveys indicating that 76% of the respondents were concerned about people observing their screens [2], while 80% admitted that they have attempted to shoulder surf the screen of a stranger [3].

The shoulder-surfing threat is worsening, as mobile devices are replacing desktop computers. More devices are mobile (over 73% of annual technical device purchases [4]) and the world's mobile worker population will reach 1.3 billion by 2015 [5]. More than 80% of U.S. employees continues working after leaving the office [6], and 67% regularly access sensitive data at unsafe locations [2]. Forty-four percent of organizations do not have any policy addressing these threats [1]. Advances in screen technology further increase the risk of exposure, with many new tablets claiming near 180-degree screen viewing angles [8].

1.2 The dangers are everywhere

Observation-based attacks to obtain displayed sensitive information can come in many forms. There are more than 3 billion camera-enabled phones in circulation [4]. Some of these devices can capture images at 40 megapixels of resolution and over 10 times optical zoom [7]. High-resolution and often insecure "security" cameras are abundant in major metropolitan areas. For example, the average resident of London is captured on CCTV over 300 times per day [9]. Finally, sensitive data can be captured by simple human sight.

Observation-based attacks can be complex. Partial images can be merged, sharpened, and reconstructed, even from reflections. Offline and cloud-based optical character recognition (OCR) solutions have only a small percentage of recognition errors, even on inexpensive low-end devices [10].

Personal information exposure can also enable other attacks, such as social engineering, phishing, and other personal identity theft threats.

1.3 The consequences can be severe

Observation-based leaks of sensitive information have led to significant personal and business losses. Recently, an S&P 500 company's profit forecasts were leaked as a result of visual data exposure [4]. In a different case, government documents were leaked when a train passenger photographed sensitive data from a senior officer's computer screen [11]. Security cameras captured the private details of Bank of America clients through windows [12]. Sensitive information relating to Prince William was captured and published because of a screen exposure to a bystander [13].

The risk of loss from shoulder surfing is also hurting business productivity. Figures show that 57% of people have stopped working in a public place due to privacy concerns and 70% believe their productivity would increase if others could not see their screen [2].

1.4 Current solutions

Techniques are available to limit the visual exposure of sensitive information. However, the focus of these systems has been limited to password entries [22, 23, 24, 25, 33, 34, 35]. Once the user has been successfully authenticated, all of the accessed sensitive information is displayed in full view. Clearly, such measures are insufficient for general computing in public when the need to access sensitive information arises. Unfortunately, many techniques used to prevent visual password leaks cannot be readily generalized beyond password protection, which motivates our work.

2. Cashtags

We present Cashtags, a system that defends against observation-based attacks. The system allows a user to access sensitive information in public without the fear of leaking sensitive information through the screen.

2.1 Threat model

We define the threat model as passive, observation-based attacks (e.g., captured video or physical observation by a human). We assume the attacker can observe both the screen of the user as well as any touch sequences the user may make on the screen, physical buttons, or keyboards. We also assume the absence of an active attack; the observer cannot directly influence the user in any way.

Although sensitive information can be presented in many forms, we focus on textual information to demonstrate the feasibility of our framework. Protecting sensitive information in other forms (e.g., images and bitmaps) will be the subject of future work.

2.2 User model

Conceptually, Cashtags is configured with a user-defined list of sensitive data items (Table 2), each with a respective Cashtags alias or a cashtag (e.g., `$visa` to represent a 16-digit credit-card number). Whenever the sensitive term would be displayed on the screen, the system displays the predefined alias instead (Fig 2.1).

Type	Actual	Alias
Name	John Smith	`$name`
Email	jsmith@gmail.com	`$email`
Username	Jsmith1	`$user`
Password	p@ssw0rd	`$pass`
Street Address	123 Main St.	`$addr`
Phone number	555-111-2222	`$phone`
Birthday	1/1/85	`$bday`
SSN	111-22-3333	`$ssn`
Credit Card	4321 5678 9012 1234	`$visa`
Account number	123456789	`$acct`

Table 2: Sample mapping of sensitive data to cashtag aliases.

Fig. 2.1: On-screen sensitive data (left) and data protected by masking with cashtag aliases (right).

At the point at which the sensitive data would be used internally by the device or app, cashtags will be replaced by their represented sensitive data items, allowing whatever login, communication, computation, transmission, or upload to proceed normally.

Cashtags also provides secure data input. A user can type in a cashtag in place of the sensitive term, permitting complex data-sensitive tasks, such as filling out a reward-card application without risk of observation from a bystander. In addition, cashtags are easier to remember than the actual sensitive data term. For example, `$visa` can be used as a shortcut for entering a 16-digit credit card number.

Users can interact with Cashtags by entering data in either alias or actual form. If the user enters the actual term, it will be converted into its corresponding alias once the full term is entered. This has the potential to inadvertently expose partial private data, an attacker could potentially see all but the last character input. In practice, auto completion is likely to expand the sensitive information within the first few characters and display it in the alias form. Entering data into the system in alias form ensures that no such partial information exposure can occur during input and is the best option to maximize protection.

2.3 Compared to password managers

The user model of Cashtags is similar to that of a password manager. To add an entry to a password manager, a user is required to key in the username and password pair. Typically, subsequent usage of the stored password involves only selecting the respective account pre-populated with the stored password. Therefore, an observer cannot see the keyed-in sequence for passwords. Similarly, Cashtags requires the user to pre-configure the system by first entering the sensitive term to be protected and the corresponding alias to represent the term. When a sensitive term is displayed, our system replaces the sensitive term with its alias without user intervention. To enter a sensitive term, the user can enter the alias, and our system will translate it into the sensitive term prior to being processed by the underlying apps.

While a password-manager-like user model provides a familiar interface, it also shares a similar threat vector of a centralized target and weakened protection in the case of physical theft. However, overcoming the shortcomings of password managers is orthogonal to the focus of this research and threat model. As research in bettering password managers advances, we can apply those techniques to enhance our system.

2.4 Design overview

Although conceptually simple, Cashtags addresses a number of major design points.

Intercepting sensitive data: Cashtags intercepts sensitive data items as they are sent to the display. For apps, this is at their common textual rendering library routines, and for users, this is at the routines that handle software keyboards and physical devices (e.g., USB and wireless input devices).

User interface: Users can type in cashtags instead of sensitive data items to compute in public. This interface allows cashtags to be compatible with existing tools such as auto completion, spellcheckers, and cut and paste. Thus, users can enter the first few characters and auto-complete the full cashtag.

Accessing sensitive data: User-entered cashtags are converted internally to the sensitive data items before the apps access the data. This way, Cashtags will not break applications due to unanticipated input formats.

Variants of data formats: Cashtags can leverage existing libraries to match sensitive data items represented in different formats (e.g., John Smith vs. John Q. Smith).

Development and deployment models: Cashtags uses a code-injection framework. This approach avoids modifying individual apps and the firmware, while altering the behavior of the overall system to incorporate Cashtags at runtime.

Cashtag repository: The mapping of cashtags to sensitive data items is stored in a password-protected repository.

3. Cashtags Design

This section will detail each Cashtags design point.

3.1 Observation-resistant approaches

We considered screen-level-masking and data-entry-tagging system design spaces prior to using the current keyword-oriented design. While all of these approaches can prevent sensitive information from being displayed, the main differences are the interception granularity and the portability of the underlying mechanisms.

Screen-level masking: One coarse-grained approach is to mask the full application window or screen regions when encountering sensitive information [61, 62]. While this approach prevents information leakage, it also prevents the user from computing using the sensitive information. For example, when encountering

an online purchase screen, the entire screen could be blurred due to sensitive information, making it unusable.

Tag-based approach: A tag-based approach requires the user to predefine the data elements as sensitive. These data elements are then tracked as they propagate through the system [16]. If a tracked data element is to be displayed on the screen, the rendering is intercepted, and the tracked data element is replaced with its corresponding alias.

This approach requires a significant system modification to support this granularity of data tracking, making it less deployable. In addition, the system resources and required accounting to track the data result in significant processing overhead incurred by the system.

Keyword-based approach: Another approach is to utilize keywords and perform pattern matching on on-screen text elements. Like the tag-based approach, this option works at the individual data element or word granularity. It also has the requirement that the sensitive data element be specified to the system prior to the point of screen rendering and subsequent visual data exposure.

The primary difference, however, is the method in which the sensitive data is identified. Rather than tracking sensitive data as it propagates through the system, this method parses data fields prior to the screen display. If a predefined sensitive element is matched, it is replaced with its alias before being rendered to the screen. We chose this approach because it achieves word granularity protection without the tag-based overhead and deployment issues.

3.2 Where to intercept sensitive data

To decide where to intercept sensitive data, we first need to understand how sensitive data traverses from apps to the screen through various display data paths. Fig. 3.1 shows the display data paths under the Android application development platform. Although different, the display data paths of iOS and Windows generally have one-to-one mappings of the components.

Window manager: A typical app displays information by invoking some user-level display or graphics library routines. Various routines eventually invoke routines in the underlying window management system (e.g., Surface Flinger for Android) before information is processed by the OS and displayed on the screen.

Arguably, the window management system might seem to be a single point at which all sensitive data can be captured. Unfortunately, by the time sensitive information arrives there, some sensitive information may have been translated into bitmaps. While OCR techniques can be used to extract sensitive text, they are still too heavyweight to be used in the display data path, which is critical for user interactions. Replacing sensitive bitmaps with non-sensitive ones would pose other obstacles we would like to avoid.

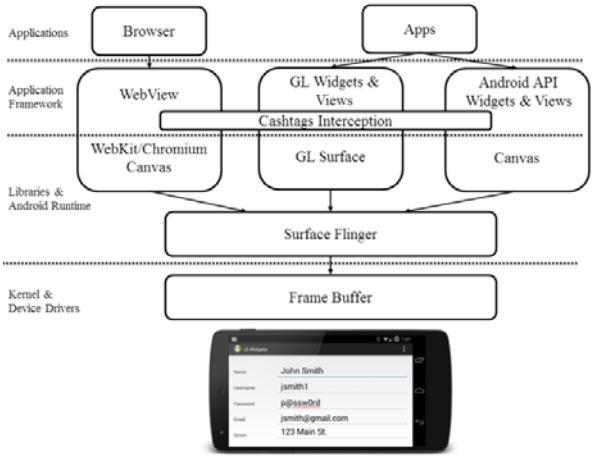

Fig. 3.1. Display data paths for the Android platform.

Applications: Another extreme is to intercept it at the app level, where the sensitive information is introduced. Potentially, we can modify a few popular, general-purpose apps (e.g., browsers) and capture most of the sensitive information. However, such solutions may tie users to specific tools. In addition, statistics show that specific app usage accounts for 86% of user time, trending away from general-purpose browsers [56]. Thus, we would need to initially modify more apps and track their updates to achieve a good coverage.

Library routines: Thus, an intermediary ground is to intercept sensitive data within a few key display and graphics library routines.

3.3 User interface

Early design: In our early user-interface design, users-defined English-like aliases in a repository to indicate sensitive data items that they wish not to be shown (e.g., use John to represent Joe). To discern these aliases when processing, we used an alternative input channel to mark them. This initial design proved problematic.

Our initial prototype was a software keyboard app with elevated privilege to offer input across applications. This implementation would be easier to port across platforms, deploy, install, and update. However, changing keyboards in the midst of a stream of input is

cumbersome in practice. This method also interacted poorly with legacy swipe-based inputs, emoticon support, auto correction, and custom dictionaries.

Further, we would need to replace the default keyboard with ours, and provide ways to switch modes between normal and sensitive entries (e.g., a screen tapping sequence). By doing so, we could retain legacy functionalities such as auto correction. On the other hand, the development effort of this approach would have been much higher, and novice users might have trouble replacing the default keyboard.

Direct input of cashtags: While there are other input interface options, the need to switch input modes to allow aliases to appear as normal text seemed superfluous (e.g., using "visa" to represent the 16-digit credit card number).

Thus, we explored the use of cashtags, where aliases are prepended with a `$` sign, to represent sensitive information. By doing so, a user can directly enter cashtags, and the mode change is encoded in the cashtag alias (e.g., use `$fname` to represent John and `$gmail` to represent jsmith@gmail.com). This method can leverage the existing custom dictionary for auto completion, which makes it easier for the user to remember and input the cashtags. This method can also utilize standard application-level development techniques, opening up the range of supported device platforms and decreasing development and installation efforts.

Direct input of sensitive information: Another supported alternative input mechanism (with some information leak) is for a user to enter the initial characters of a sensitive data item. As soon as the auto completion detects that, Jo is likely to mean Joe, for example, it will be automatically masked with `$john`. The user then can choose `$john` and proceed.

Additional Cashtags semantics: Recursion is supported, so we can use `$signature` to represent `$fname $lname $gmail`, which in turn maps to John Smith, jsmith@gmail.com. We disallow circular cashtags mappings (e.g., use `$john` to represent `$joe`, and `$joe` to represent `$john`).

3.4 Accessing sensitive information

One design issue addresses converting cashtags back to the sensitive data for access by apps. Normally, when an app wants to access the sensitive information and to send it back to the hosting server, we must make sure that the conversion is performed prior to access, so that the app would never cache, store, or transmit the cashtags. The concern is that cashtags may break an app due to the violation of the type or formatting constraints.

We also must make sure that the cashtags are actually entered by the user, not just pre-populated by the app. Otherwise, a malicious app can extract sensitive information just by displaying cashtags.

There are certain exceptions where it is desirable to operate directly on cashtags instead of the sensitive information. For example, the auto-completion task will auto complete cashtags (`$fn` to `$fname`), not the sensitive information it represents. By doing so, the handling of text span issues is simplified because cashtags usually differ in text lengths when compared to the sensitive information they represent.

3.5 Variants of data formats

Sensitive data may be represented in multiple formats. For example, names can be represented as combinations of first, last, and middle initials (e.g., John Smith; John Q. Smith). Accounts and social security numbers can be represented using different spacing and/or hyphenation schemes (e.g., 123456789; 123-45-6789). Fortunately, we can leverage existing regular expression libraries (`java.util.regex.*`) to perform such matching.

Another issue involves the type restriction of the input field. For example, a number field (e.g., SSN) may prevent the use of cashtags (`$ssn`). To circumvent these restrictions, we allow users to define special aliases (e.g., 000-00-0000) to represent certain types of sensitive information (e.g., social security numbers).

3.6 Deployment and development models

To avoid modifying individual applications, we considered two options to provide system-level changes: (1) custom system firmware images (ROMs) and (2) code-injection frameworks (e.g., Android Xposed)

By utilizing a custom system firmware image, complete control of the operating system is provided. (This approach assumes that the full source is available.) In addition, ROM-based solutions can offer a more unified testing environment. However, the changes would be restricted to device-specific builds. Only hardware for which the source is built would have access to the modified system. This also limits user preference by restricting use only for a specific system image. It would additionally require regular maintenance, and it would break vendor over-the-air update functionality.

Instead, we used a code-injection framework, which overrides library routines and incorporates our framework into execution prior to the starting of apps.

Code injection offers streamlined development, as standard application development tools can be used. In addition, these modules can be distributed and deployed as applications. Because code injection only relies on the underlying system using the same set of libraries, the resulting system is more portable and less coupled to versions and configurations of system firmware.

The installation and use of the code-injection framework requires root access to the device. This is, however, not a firm requirement and exists only for this prototype; Vendors and OEMs can incorporate Cashtags into system firmware providing the same functionality without exposing root. This deployment model is advisable to further enhance security for a production system.

3.7 Cashtags app and repository

Cashtags aliases and sensitive data items are maintained in a repository. The Cashtags app coordinates the interactions between various apps and the repository. The app also provides password-protected access to add, edit, remove, import, and export sensitive terms and corresponding cashtags.

Cashtags provides per-application blacklisting, excluding specific applications from being code-injected (or activated) with cashtag-replacement code. For example, the cashtag repository itself must be excluded due to circular dependencies. To illustrate, suppose a cashtag entry maps `$fname` to Joe. If Cashtags is enabled, the screen will show that `$fname` is mapped to `$fname`; when saved, Joe will be mapped to Joe. Apps with a low risk to reveal sensitive information can be excluded for performance issues (e.g., games, application launchers, home screens).

4. Implementation

We prototyped Cashtags on the open-source Android platform. Our code-injection framework allows Cashtags to operate on any Android device with the same display and graphics libraries and root access. This section will first detail the Android display data paths, explain the code-injection framework, and discuss how various display data paths are intercepted and how cashtags are stored.

4.1 Android display elements

Fig 3.1 has shown a top-level view of the various ways Android apps and browsers display information on the screen. This section provides further background on Android terminologies. Corresponding terminologies for text widgets on Apple and Windows devices are listed in Table 4.

	Android	Apple	Windows
Text Labels	TextView	UITextView	TextBlock
OpenGL Text	GLES20 Canvas	GLKView	Direct3D
Editable Text	TextView	UITextView	TextBlock
Webapp Text	WebView	UIWebView	WebView
Browser/ WebView	WebView	UIWebView	WebView

Table 4: Widget terminologies on Android, Apple, and Windows platforms.

The Android display is composed of views, layouts, and widgets. `View` is the base class for all on-screen user interface components.

Widgets: The term widget is used to describe any graphic on-screen element. Different widgets can be used to display static text labels (e.g., `TextView`), user input boxes (e.g., `EditText`), controls (e.g., `Buttons`), and other media (e.g., `ImageView`).

`Views` are organized into `ViewGroups`, the base class for all screen layouts. `Layouts` are arrangements of views within vertical or horizontal aligned containers (e.g., `LinearLayout`), or arranged relative to other views. Nesting of `ViewGroups` and `Layouts` allows complex custom composites to be defined.

Collectively, this tree of layouts and widgets forms a view hierarchy. When the screen is drawn, the view hierarchy is converted from logical interface components into a screen bitmap. Fig. 4.1 shows a simple user input form and its composition of various widgets and layouts.

Text rendering: Text can be rendered on the screen through several mechanisms (Fig 3.1), most frequently the `TextView` widget. An `EditText` is an extension of the `TextView` that provides an interface for text input from the user via the on-screen software keyboard, hardware keypads, voice input, and gestures. These widgets can be pre-filled with text by the app internally or through suggestion or auto-correction interfaces.

Text can also be rendered on screen via `OpenGL Canvas`, other graphic rendering libraries, browser rendering engines, and other methods, many of which do not inherit from the base `TextView` widget. This plethora of methods complicates practical capture of all possible ways text could get onto the screen.

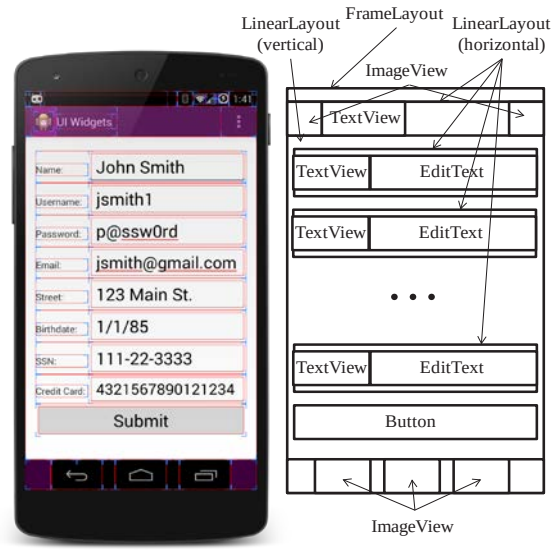

Fig. 4.1. Decomposition of on-screen views, layouts, and widgets of a simple app input form.

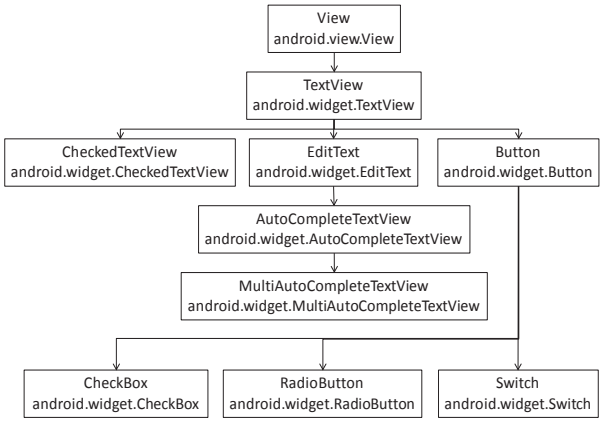

Fig. 4.2. Simplified inheritance hierarchy of Android on-screen views and widgets.

4.2 Android code-injection framework

Cashtags uses the Android Xposed code-injection framework. The development cycle is accelerated by short-circuiting the need to perform complete device firmware rebuilds from scratch.

Underneath the hood, whenever an app is started, Android forks off a new virtual machine. The Android Xposed framework allows overriding library routines to be inserted into the Java `classpath`, prior to the execution of the new virtual machines. Thus, the overall system behavior is altered without modifying either the apps or the underlying firmware.

Individual class methods can be *hooked*, allowing injected code to be executed prior to, following the completion of, or in place of the base-method calls. Private or protected member fields and functions can also be accessed and modified, and additional fields or functions can be added to the base class or object granularity.

4.3 Sensitive data intercepting points

With the background of the Android display data paths (Fig. 3.1) and the code-injection framework, we can determine where and how to intercept sensitive information. Since all text-displaying screen widgets are descendants of the `TextView` class (Fig. 4.2), we only needed to hook `TextView` (`android.widget.TextView`) to intercept all widgets containing static sensitive text. For user input, we hooked `EditText` (`android.widget.EditText`) to capture sensitive data or cashtags entered via on-screen software keyboards, (integrated, plugged, or wirelessly connected) hardware keypads, voice input, and gestures. For display through the OpenGL libraries, we intercepted GLText (`android.view.GLES20Canvas`). For browsers, we intercepted `WebView` (`android.WebKit/WebView`).

4.4 `TextView`

Fig. 4.3 shows a simplified version of the implementation of the `TextView` widget in the Android API, present since version 1 of the Android SDK. The `getText`() and `setText`() methods of the `TextViews` are hooked and modified (the `setText`() method in `TextView` is inherited by `EditText`, to be detailed later). We also added `mAlias` to map the sensitive text to the corresponding cashtag.

4.5 `EditText`

In addition to the functionality inherited from `TextView`, `EditText` must also handle additional actions that can be performed by the user, app, or system to modify on-screen text.

```
public class TextView extends View
  implements ViewTreeObserver.
  OnPreDrawListener {
    ...
    private CharSequence mText;
    private CharSequence mAlias:
    ...
    public CharSequence getText() {
      return mText;
    }
    ...
    private void setText(CharSequence
     text, BufferType type, boolean
     notifyBefore, int oldlen) {
      ...
      mBufferType = type;
      mText = text;
    }
    ...
  }
```

Fig. 4.3. Simplified TextView implementation. Bolded functions `getText()` and `setText()` are hooked and modified. An additional private field `mAlias` is added for mapping to a displayed cashtag, if applicable.

For cases where the system or app has pre-populated a text box with input, the `TextView` injection handles the cashtag replacement. Since the `EditText` class extends from the `TextView` base class, this functionality is provided through inheritance. This is also the case for nearly every other on-screen text widget, which also descend from the base `TextView`.

Fig. 4.4 and Fig. 4.5 show how Cashtags interacts with `TextView` and `EditText` objects. When these `getText()` and `setText()` methods are called by the app or through system processes, such as auto correct or to be rendered on screen, Cashtags will determine whether to return the alias or the sensitive data, depending on the caller.

User input can be entered through software keyboards or through physical devices. In both cases, Cashtags operates similar to, and through the same interface, as the auto-correct service. This `TextWatcher` (`android.text.TextWatcher`) interface handles events when on-screen text has been modified. `EditTexts` internally maintain an array of these `TextWatcher` event handlers. Cashtags, as one of these handlers, is activated after any character is modified within the text field.

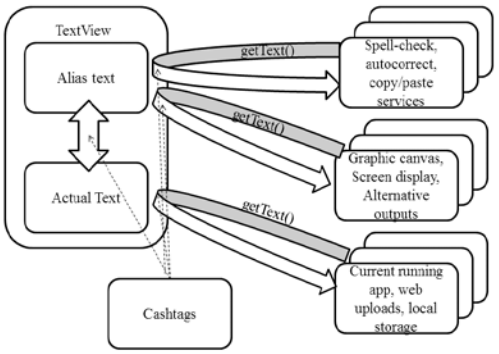

Fig. 4.4. Interactions among Cashtags, `TextView`, and other software components. `getText()` returns either the cashtag or actual text depending upon the service making the request.

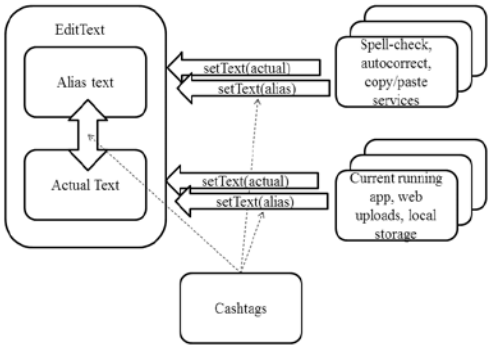

Fig. 4.5. Interactions among Cashtags, `EditText`, and other software components. `setText()` returns either the cashtag or actual text depending upon the service making the request.

This functionality is also achieved through the view `OnFocusChangeListener` (`android.view.OnFocusChangeListener`). This event handler works at the granularity of the full text field rather than the individual characters of the `TextWatcher`. This is more efficient, since the text replacement only occurs once per text field. It does, however, risk additional on-screen exposure of sensitive information, since direct input of actual sensitive terms would remain on-screen as long as the cursor remains in that text field. The input of the cashtag alias does not have this risk and further reduces any partial exposure during term input.

In both cases, the constructor of the `EditText` class is hooked and the corresponding `OnFocusChangeListener` or `TextWatcher` object is attached at runtime. User settings allow activation of either or both interception methods.

4.6 `OpenGLES Canvas`

The implementation for `OpenGLES Canvas` is similar to the base `TextView` only with different parameter types. The only distinction is that no accompanying `getText`() equivalent is present in this object, so no manipulation is necessary beyond `drawText`().

4.7 `WebView`

Unlike the previous screen widgets, the relevant interception points for screen rendering for WebKit or Chromium browser engines are all below the accessible Android/Java layer. Thus, they cannot be code-injected through the same mechanisms used for previous screen widget cases. We attempted custom compilations of the browser engines with similar widget display interception, but abandoned the effort for portability concerns.

Instead, `WebView` interception is handled similarly to a web browser plug-in. This decision maintains the portability goal of the system design.

Cashtags intercepts web rendering immediately before it is first displayed on-screen. The HTML is pre-processed with JavaScript to extract the DOM. Cashtags iterates over the text nodes and makes the appropriate text replacements of sensitive data to corresponding cashtags.

Other options were explored using specific browser and proxy requests through the web server. However, all apps that use cross-platform frameworks (PhoneGap, Apache Cordova, JQuery Mobile, etc.) run locally and could not easily be piped though this service. For this reason, we favored the plug-in approach over other alternatives.

4.8 Cashtags repository

Sensitive terms are stored as encrypted `SharedPreference` key/value pairs, which uses the AES encryption from the Java Cryptography Architecture (`javax.crypto.*`). This structure is accessed by enabled apps through the `XposedSharedPreference` interface.

5. Evaluation Methods

Cashtags was evaluated for how well the intercepted points prevent specified information from being displayed on the screen, verified by screen captures and OCR processing. This coverage was measured by enumerating common paths sensitive text can traverse to the screen. We also evaluated user input through popular apps, making sure that the cashtags reverted to the sensitive data items when accessed by the apps. Finally, we evaluated the performance overhead of Cashtags.

5.1 API coverage evaluation

For Android API coverage, we focus on the `TextView` and `EditText` display data paths, which account for 86% of usage hours for mobile devices [56], though Cashtags can also handle the remaining 14% of browser-based access. The selected sensitive information (Table 2) is based on the Personally Identifiable Information (PII) specified by the U.S. government and NIST standards [59]. We enumerate all combinations of the input phrase type (e.g., numbers, strings), case sensitivity, common widget, layout, theme, and lifecycle for these data paths. Each combination is used to demonstrate that the PII terms are not displayed on the screen from the app, as user input of the sensitive data, or as user input of the cashtag alias. In all three cases, we also demonstrate that the PII term is correctly returned from Cashtags when used by the app.

This totals 1,728 tests for the static text widgets and inputs, with 526 additional cases for widgets that permit user input via software keyboards as well as physical devices (on-board hardware, USB, or wireless input devices). Table 5.1 shows the full list of configurations.

For each combination, the Android Debug Bridge [60] and UIautomator tool [36] are used to capture the device layout view hierarchies and screenshots. The contents of the actual and cashtag fields are compared for conversion correctness. The device screenshot is processed using Tessseract OCR [21] to confirm if the actual PII term has been properly masked on the screen.

For each combination, we also demonstrate that both text input as a sensitive term and cashtag are correctly converted to the sensitive term when accessed by the app. Since the access of sensitive data within the app normally involves remote actions, we also emulated this scenario and performed remote verification. Once screen processing is completed, the app accesses the text fields and uploads them to Google Sheets/Form. The uploaded sensitive items and cashtag submissions are compared for accuracy based on expected values.

Our results show that Cashtags behaves correctly for all test cases. For each test case, Cashtags identified input containing sensitive data in both the actual and cashtag form, prevented the display on the screen of the sensitive term, and determined correctly when to convert back to the sensitive data.

Input phrase type (4): Alphabetic phrase, numeric phrase, alphanumeric phrase, Alphanumeric with symbols.
Phrase case (2): Case Sensitive Text, Case In-sensitive Text
Widget type (9): `TextView` (`android.widget.TextView`), `CheckedTextView` (`android.widget.CheckedTextView`), `Button` (`android.widget.Button`), `CheckBox` (`android.widget.CheckBox`), `RadioButton` (`android.widget.RadioButton`), `Switch` (`android.widget.Switch`), `EditText` (`android.widget.EditText`), `AutoCompleteTextView` (`android.widget.AutoCompleteTextView`), `MultiAutoCompleteTextView` (`android.widget.MultiAutoCompleteTextView`)
Layout type (2): `LinearLayout` (`android.widget.LinearLayout`), `RelativeLayout` (`android.widget. RelativeLayout`)
Theme type (3): Default theme, System theme, User-defined theme.
Generation method (2): Static XML, Dynamic Java
Lifecycle type (2): Activity-based lifecycle, Fragment-based lifecycle

Table 5.1: Android API test combinations.

5.2 App coverage evaluation

The Google Play market has more than one million of published applications accessible by thousands of different hardware devices [70], making the enumeration of all possible users, devices, and application scenarios infeasible. Thus, we chose a representative subset of popular apps to demonstrate coverage of Cashtags. Categorically, these application types are email, messaging, social media, cloud and local storage, office, and finance. Table 5.2 shows the selected apps, arranged according to these categories. These apps were selected using download metrics from the Google Play marketplace, excluding games and utility apps for lack of relevance in terms of displaying sensitive data on screen. The presence of a form of external verification was also used in the application selection. Apps typically bundled with mobile devices were also tested for correct operation.

Email: AOSP Email, Gmail, K9 Mail: User reads an email containing a sensitive term in its cashtag form. A Cashtags-enabled system should display the email with two instances of the cashtag. User composes an email with a sensitive term and its cashtag. Remote system not running Cashtags should display email with two sensitive term instances.
Messaging: Messaging, Hangouts, Snapchat: User reads a message containing a sensitive term and cashtag. A Cashtags-enabled system should display the message containing two instances of the cashtag. User composes a message with a sensitive term and its cashtag. A remote system not running Cashtags should receive the message with two sensitive term instances.
Social: Facebook, Twitter, Google+: User reads text with a sensitive term and its cashtag from a tweet/post/update. A Cashtags-enabled system should display the tweet/post/update with two instances of the cashtag. User composes a new tweet/post/update with a sensitive term and its cashtag. A remote system not running Cashtags should receive the tweet/ post/update with two sensitive term instances.
Storage: Dropbox, MS OneDrive, File Manager: User opens an existing file containing a sensitive term and its cashtag. A Cashtags-enabled system should display the file containing two instances of the cashtag. User creates a file with a sensitive term and its cashtag. A remote system not running Cashtags should display file with two sensitive term instances.
Office: GoogleDocs, MS Office, QuickOffice: User reads a document containing a sensitive term and its cashtag. A Cashtags-enabled system should display the file with two instances of the cashtag. User creates a document containing a sensitive term and its cashtag. Remote system not running Cashtags should see two sensitive term instances.
Finance: Google Wallet, Paypal, Square: User reads a document containing a sensitive term and its cashtag. Cashtag-enabled system should display the document with two instances of cashtag. User creates a document containing a sensitive term and its cashtag. A remote system not running Cashtag should see two sensitive term instances.

Table 5.2: Per-category app test tasks.

The operation performed on each is based on a commonly performed use case or task for each category. Table 5.2 shows the operation performed for each category and respective app.

Our results show that Cashtags behaves correctly for 97% of task and app combinations, except the MS Office Mobile tests. The reason is because of the custom View (`docRECanvasHost`) used for the primary user interaction, which is not a descendant of an `EditText`. Thus, our system does not intercept it. All of the other apps tested have user input through an `EditText`, or a custom class inheriting from an `EditText`. This exception, as well as other custom views could be made to work with Cashtags using case-specific handling for internal functions and parameters that map to the equivalent `EditText` functions.

5.3 Overhead

Regarding overhead, we measured the incremental lag Cashtags added to the system. We ran a modified version of the Android API coverage test (Section 5.1) with and without Cashtags enabled. The screenshots, layout hierarchy dumping, and all other non-essential automation elements were removed prior to the test execution. The test execution durations are compared, and the additional lag introduced by the system was calculated. This test was run with and without the remote data verification to determine the effects of network lags on our system overhead.

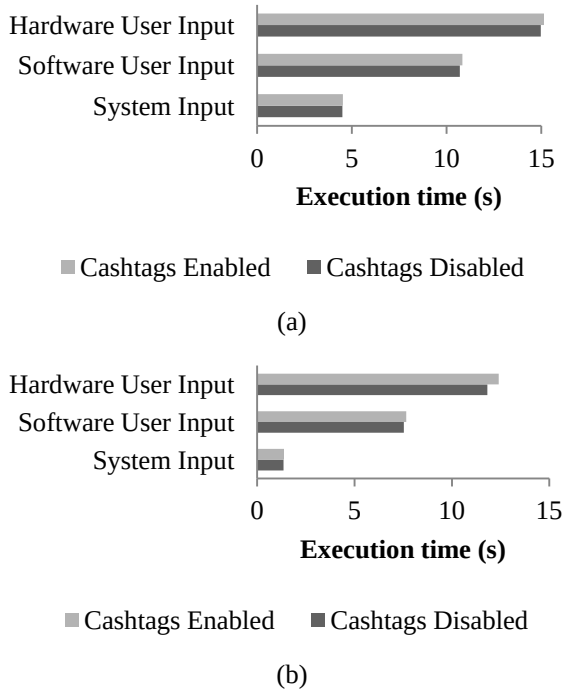

Fig. 5.1. Comparison of mean app task execution time with and without Cashtags enabled, using system, software and hardware text input (a) with and (b) without web request for tests. Hardware refers to input from physically or wirelessly connected hardware keyboard and software refers to input via on-screen software keyboard.

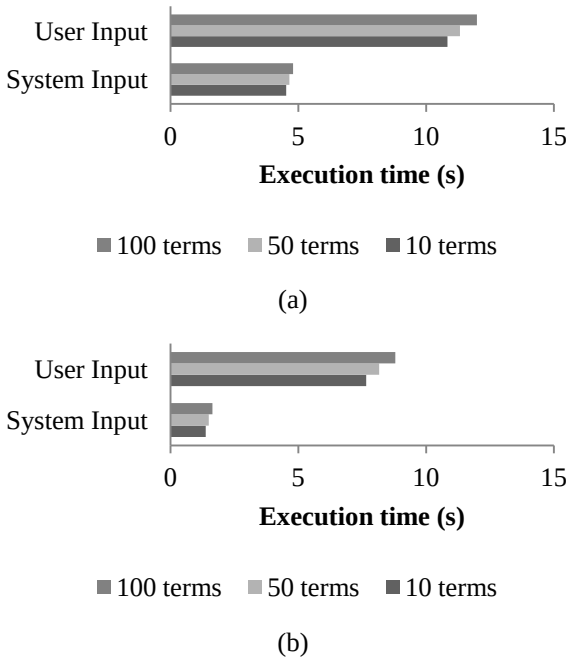

Fig. 5.2. Comparison of mean app task execution time with an increasing number of cashtag entries, using system and user inputs (a) with and (b) without web request for tests.

Fig. 5.1(a) shows the Cashtags system incurs an average 1.9% increase in application runtime. For tests including remote verification, Cashtags incurred an average of 1.1% increase over the baseline tests. For tests excluding the time consuming remote verification, Fig. 5.1(b) shows that Cashtags incurred an average of 2.6% over baseline. Therefore, the additional overhead of Cashtags would not be perceivable to the user.

Testing was repeated using 50 and 100 items, which is significantly more than the list of terms specified by PII. Fig. 5.2 show that the performance degrades linearly as the number of cashtags entries increases. However, we can easily replace the data structure to make the increase sublinear.

Cashtags is additionally evaluated for boot-time overhead. Changes to the Cashtags repository currently require a reboot to take full effect. While this operation is not in the critical path, the overhead is relevant. The results of the boot lag are shown in Fig. 5.3.

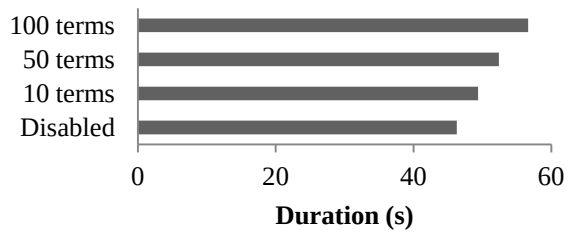

Fig. 5.3. Comparison of device startup times with varying number of cashtag entries and with system disabled.

5.4 Usability overhead

To demonstrate the usability of Cashtags, we calculated the configuration and usage overhead seen by a typical user working on typical data. We used common sensitive data elements with known character lengths (such as credit card numbers) as well as those for which published data on average term length was available (such as common names) [63, 64].

For other elements without known average length availability, we substituted a typical minimum accepted value. Table 5.3 shows the comparison of these fields against the suggested cashtag alias length.

In nearly every case, the cashtag alias is significantly shorter than the average length of the sensitive data element. On a keystroke count basis, the amortized effort of the initial configuration can be overcome with only two or three uses of the Cashtag alias. Longer names and emails require additional keystrokes for initial configuration but yield greater keystroke savings for each time the data element is entered. In addition, the aliases in the table are based on the suggested terms for ease of user recall; even shorter terms could be substituted to reduce additional data entry overhead.

5.5 Quantification of the time savings for an end user

The real efficiency savings for Cashtags is the ability to represent hard-to-recall terms (e.g., account numbers) with easy-to-recall aliases. The user simply assigns the sensitive data a memorable alias and can refer to them.

This also adds convenience when sensitive data changes. For example, consider the case of a stolen credit card. Once the replacement is issued, the user need only to update the underlying data element, continuing to use the alias without the need to memorize a new number. In some cases of personal information change, the defunct data could still be considered sensitive and be prevented from being displayed on the screen. In such cases, the defunct data element could be assigned to a new alias. For example, consider a street address change: the alias `$street` is assigned to the new data, and the past street address can be assigned to `$street_old`.

Type	Actual	Alias	Alias	Diff
First Name	6	`$fname`	6	0
Last Name	6	`$lname`	6	0
Email	20	`$email`	6	14
Username	9	`$user`	5	4
Password	9	`$pass`	5	4
Phone number	10	`$cell`	5	5
Birthday	10	`$bday`	5	5
SSN	9	`$ssn`	4	5
Credit Card	16	`$visa`	5	11
Acct. number	12	`$acct`	5	7

Table 5.3: Typical keystroke counts for common sensitive private data terms [63, 64] and corresponding suggested Cashtag alias.

6. Related Work

Previous works include both systems that secure against observation-based attacks and those that provide similar privacy protection over network channels.

6.1 Visual authentication protection

Prior work on protection against visual exposure is focused on securing the act of authentication. By far the earliest is the technique of Xing out or not printing entered passwords on login screens [15]. Most others can be generalized as an augmentation or replacement of password entry mechanisms.

Password managers: Password managers allow the user to select a predefined username and password pair from a list for entry into the login fields [14]. This also allows a user to use different passwords for different apps without the need to remember each individually.

Hardware-based authentication: Hardware-based authentication techniques utilize specialized USB dongles [17], audio jacks [18], short-range wireless communication [19], or Bluetooth connections [20] to connect to the authenticating machine. Therefore, a bystander cannot obtain the authentication information by observation.

Graphical passwords: Graphical passwords or Graphical User Authentication (GUA) [22] replace the alpha-numeric password with a series of images, shapes, and colors. The user needs to click a sequence of

human faces [23], object sequences as part of a story [24], or specific regions within a given image [25].

Biometrics: Biometric authentication mechanisms augment password entry (something one knows) with a feature unique to one's personal biology (something one is). The most commonly used of these biometric identifiers includes contours of the fingerprints [26], iris and retinal configuration of the eye [27], and geometries of the face [28] and hand [29]. Behavioral characteristics such as keystroke latency [30], gait [31], and voice [32] can also be used.

Gesture-based authentication: Gesture-based authentication techniques allow the user to perform specific tap [33], multi-finger presses [34], or swipe sequences on-screen [35] to represent a password.

Cognitive challenges and obfuscation: Other techniques have attempted to make games of the authentication procedure [37]. These techniques utilize challenge-response questions and cognitive tasks to increase the difficulty of the login session [38]. Other techniques use obfuscation (e.g., the hiding of cursors [39], confusion matrices [40], and recognition [41]) rather than recall-based methods, to confuse onlookers.

Alternative sensory inputs: Some systems utilize audio direction [42] or tactile and haptic feedback from the vibration motors on devices [43] to provide the user with the appropriate cue for the necessary response. The user then responds with the passphrase corresponding to the cue via traditional input methods.

There are also systems that extend GUAs by requiring sequential graphical inputs and use mechanics, such as eye tracking, blinking and gaze-based interaction for the user to input the graphical sequence [44]. Systems have even demonstrated the capability of using brain waves for this task; a user may only need to think a specific thought to authenticate with a system [45]. These methods are also useful alternatives for authentication of people with visual or audio sensory disabilities [46].

6.2 Physical barriers and screen filters

Physical barriers can serve as a means of limiting the amount of screen exposure to bystanders. This can be as simple as office cubicles. However, they are clearly not suitable for the increasingly mobile modern workforce. Other solutions, such as the 3M Privacy Filter [67] approach the problem by limiting the field of view of the screen. This may serve to reduce exposure, but larger screens are still visible for a larger area and can be seen by unauthorized viewers directly behind the device.

The Lenovo Sun Visor [68] and Compubody Sock [69] reduce the screen visibility further by completely blocking out all non-direct visual exposure. However, this also blocks the user's own field of view, leaving them susceptible to external threats such as pick pocketing.

6.3 Wearable devices

Wearable headsets such as Google Glass [57] prevent screen exposure by moving the screen directly in front of the user's eyes. However, the current generation devices have limited application support. In addition, much of the user input is performed by audio cues, which translates the visual sensitive data leaks to audio ones.

Oculus Rift [65] and Samsung Galaxy Wearable [66] permit similar private screen viewing. However, they currently do not permit general-purpose computing. Additionally, like physical barriers, these devices block the user's field of view, increasing the vulnerability to external physical threats.

6.4 Digital communication channel protection

Many protocols and systems have been developed to handle other aspects of privacy-oriented attacks through the encryption of the digital communication channel. Transport Layer Security and Secure Sockets Layer can enhance security by providing session-based encryption [47]. Virtual Private Networks can be used to enhance security by offering point-to-point encryption to provide secure resources access across insecure network topologies [48]. Proxy servers [49] and onion routing protocols such as Tor [50], can add extra privacy by providing obfuscation of the location and anonymization of IP addresses.

Other solutions have been developed to enhance privacy of browsers. Do-not-track requests can be included in the HTTP headers to request that the web server or application disable its user and cross-site tracking mechanisms [51]. Many browser extensions and plug-ins exist to block ads [52], analytics, beacons, and other tracking mechanisms [53]. Other systems alert the user when specific sensitive elements are leaked [54]. They prevent the transmission of sensitive data without explicit user permission [55], and the cryptography secures access to sensitive data outside of trusted situations [16].

6.5 Compared to Cashtags

Despite the various mechanisms mentioned, the visual channel remains largely open. A limited number of tools are available to obfuscate sensitive data other than during the act of authentication. Other tools developed

for data encryption are not originally designed for such purposes.

Password-based solutions and biometrics are effective in handling visual leaks during the act of authentication, but they cannot be generalized to handle other cases. No existing mechanism is in place to allow arbitrary data to be marked as sensitive. To our best knowledge, Cashtags is the only system that can protect general data from shoulder surfing.

7. Discussion and Limitations

Increased coverage: Cashtags widget-level text manipulation works for apps that use standard text-rendering methods. Should developers deviate from such standards, Cashtags would not capture such cases. Still, the additions required to incorporate these custom methods to work within Cashtags would be minimal if knowledge of the custom text display functions and parameters were provided.

Cashtags currently is optimized for coverage rather than performance. Thus, one future direction is to explore better text-processing methods and data structures.

Common names: Commonly occurring names can be problematic. Consider a user John Smith, with Cashtag aliases of his name: John→`$fname`, and Smith→`$lname`. Therefore, all on-screen instances of John are masked as `$fname`. If John opens his browser and Googles the name John Travolta, all returned search results would be displayed with on-screen representations as `$fname` Travolta. If an on-looker was able to observe the above search queries, and was aware of the operation of Cashtags, he or she might be able to derive the sensitive data from the context. This limitation is isolated to common phrases; numerical phrases would be less subject to this issue.

Data formatting: For data formatting and types, many cases are handled through transformations of text fields, including the removal of spaces and symbols, and capitalization mismatches. However, data that expands across `TextViews` is not recognized (e.g., input fields for a credit card split into parts rather than combined into a single field). Cashtags could handle this if each part of the credit-card number were added to the repository.

Handling business use cases: This paper presents Cashtags in light of protecting personal sensitive information. However, with more advanced cashtag mapping rules internally implemented via regular expression templates, we can extend our framework to handle business use cases. For example, for banking account apps, we can mask all of the dollar amounts to `$#`, with a fixed number of digits. A user can specify common preset amounts using cashtags (e.g., pay `$rent` to `$apt_acct`). For database apps, we can mask fields with specific template formats (e.g., phone numbers, identification numbers with certain prefixes). While such extension will require ways to specify app-specific rules, our core framework remains the same.

Generalization of approach: The Cashtags system was prototyped on Android. However, the general approach of screen rendering and user input interception can easily be generalized.

Human subject study: One aspect that is important to system usability is the frequency that sensitive data is entered or displayed. The actual utility of Cashtags is directly related to how frequently personally identifiable information is accessed and input by the user. Unfortunately, to our best knowledge statistics on the frequency of such accesses are not available. A future human subjects study of Cashtags can help determine this frequency of sensitive data access, as well as further evaluate system effectiveness and usability.

8. Conclusion

Cashtags is a first step toward protection against visual leaks of on-screen data. The system demonstrates that it is possible to perform most mobile computing tasks in public locations without exposing sensitive information. The evaluation of the system shows that this can be accomplished efficiently, with minimal perceived overhead. The app coverage test confirms that the system handles general-purpose tasks and maintains full functionality with nearly all tested common use cases. These results suggest that Cashtags will likely work on most other mobile apps, providing unified, device-wide protection against shoulder surfing.

9. Acknowledgements

We would like to thank our shepherd Tadayoshi Kohno, and anonymous reviews for their invaluable feedback. This work is sponsored by NSF CNS-1065127. Opinions, findings, and conclusions or recommendations expressed in this document do not necessarily reflect the views of the NSF, FSU, UCLA, or the U.S. government.

References

[1] Honan, Brian. “Visual Data Security White Paper”, July 2012. BH Consulting & European Association for Visual Data Security. http://www.visualdatasecurity.eu/wp-

content/uploads/2012/07/Visual-Data-Security-White-Paper.pdf. Retrieved 4/2014

[2] Thomson, Herbert H, PhD. "Visual Data Breach Risk Assessment Study." 2010. People Security Consulting Services, Commissioned by 3M. http://solutions.3m.com/3MContentRetrievalAPI/BlobServlet?assetId=1273672752407&assetType=MMM_Image&blobAttribute=ImageFile. Retrieved 4/2014

[3] Vikuiti Privacy Filters. "Shoulder Surfing Survey". 2007. Commissioned by 3M UK PLC. http://multimedia.3m.com/mws/mediawebserver?6666660Zjcf6lVs6EVs66SlZpCOrrrrQ-. Retrieved 4/2014

[4] European Association for Visual Data Security. "Visual Data Security", March 2013. http://www.visualdatasecurity.eu/wp-content/uploads/2013/03/Secure-Briefing-2013-UK.pdf. Retrieved 4/2014

[5] International Data Corporation. "Worldwide Mobile Worker Population 2011-2015 Forecast." http://cdn.idc.asia/files/5a8911ab-4c6d-47b3-8a04-01147c3ce06d.pdf. Retrieved 4/2014

[6] Good Technology. "Americans are Working More, but on their Own Schedule", July 2012. http://www1.good.com/about/press-releases/161009045.html. Retrieved 4/2014

[7] Nokia, USA. "Nokia Lumia 1020", http://www.nokia.com/us-en/phones/phone/lumia1020/. Retrieved 4/2014

[8] NPD DisplaySearch. "Wide Viewing Angle LCD Technologies Gain Share Due to Tablet PC Demand". January 2012. http://www.displaysearch.com/cps/rde/xchg/displaysearch/hs.xsl/120119_wide_viewing_angle_lcd_technologies_gain_share_due_to_tablet_pc_demand.asp. Retrieved 4/2014

[9] Pillai, Geetha. "Caught on Camera: You are Filmed on CCTV 300 Times a Day in London", International Business Times, March 2012. http://www.ibtimes.co.uk/britain-cctv-camera-surveillance-watch-london-big-312382. Retrieved 4/2014

[10] Loh Zhi Chang and Steven Zhou ZhiYing. "Robust pre-processing techniques for OCR applications on mobile devices", In Proceedings of the 6th International Conference on Mobile Technology, Application & Systems (Mobility '09). ACM, New York, NY, USA, Article 60 , 4 pages. DOI=10.1145/1710035.1710095 http://doi.acm.org/10.1145/1710035.1710095

[11] Owen, Glen. "The zzzzivil servant who fell asleep on the train with laptop secrets in full view", November 2008. http://www.dailymail.co.uk/news/article-1082375/The-zzzzivil-servant-fell-asleep-train-laptop-secrets-view.html. Retrieved 4/2014

[12] Penn, Ivan. "Simple fix to bank security breach: Close the blinds", Tampa Bay Times. December 2010. http://www.tampabay.com/features/consumer/simple-fix-to-bank-security-breach-close-the-blinds/1139356. Retrieved 4/2014

[13] Davies, Caroline. "Prince William photos slip-up forces MoD to change passwords", The Guardian, November 2102. http://www.theguardian.com/uk/2012/nov/20/prince-william-photos-mod-passwords. Retrieved 4/2014

[14] J. Alex Halderman, Brent Waters, and Edward W. Felten. "A convenient method for securely managing passwords", In Proceedings of the 14th international conference on World Wide Web (WWW '05). ACM, New York, NY, USA, 471-479. DOI=10.1145/1060745.1060815 http://doi.acm.org/10.1145/1060745.1060815

[15] CTSS Programmers Guide, 2nd Ed., MIT Press, 1965

[16] C. Fleming, P. Peterson, E. Kline and P. Reiher, "Data Tethers: Preventing Information Leakage by Enforcing Environmental Data Access Policies," in International Conference on Communications (ICC), 2012.

[17] Yubico, Inc. "About YubiKey", 2014. http://www.yubico.com/about. Retrieved 4/2014

[18] Square, Inc. "About Square", 2014. https://squareup.com/news. Retrieved 4/2014

[19] Google, Inc. "Google NFC YubiKey Neo", September 2013. http://online.wsj.com/news/articles/SB10001424127887323585604579008620509295960

[20] Wayne Jansen and Vlad Korolev. "A Location-Based Mechanism for Mobile Device Security", In Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering (CSIE '09), Vol. 1. IEEE Computer Society, Washington, DC, USA, 99-104. DOI=10.1109/CSIE.2009.719 http://dx.doi.org/10.1109/CSIE.2009.719

[21] Google, Inc. Tesseract-OCR. https://code.google.com/p/tesseract-ocr/

[22] Blonder, Greg E. "Graphical Passwords". United States patent 5559961, Lucent Technologies, Inc. 1996.

[23] Passfaces Corporation. "The Science Behind Passfaces", June 2004. http://www.realuser.com/published/ScienceBehindPassfaces.pdf

[24] Darren Davis, Fabian Monrose, and Michael K. Reiter. "On user choice in graphical password schemes", In Proceedings of the 13th conference on USENIX Security Symposium - Volume 13 (SSYM'04), Vol. 13. USENIX Association, Berkeley, CA, USA, 11-11.

[25] Susan Wiedenbeck, Jim Waters, Jean-Camille Birget, Alex Brodskiy, and Nasir Memon. "PassPoints: design and longitudinal evaluation of a graphical password system" International Journal of Human-Computer Studies. 63, 1-2 (July 2005), 102-127. DOI=10.1016/j.ijhcs.2005.04.010 http://dx.doi.org/10.1016/j.ijhcs.2005.04.010

[26] Jain, A.K.; Hong, L.; Pankanti, S.; Bolle, R., "An identity-authentication system using fingerprints," Proceedings of the IEEE, vol.85, no.9, pp.1365, 1388, Sep 1997. doi: 10.1109/5.628674

[27] J. Daugman. "How iris recognition works", IEEE Transactions on Circuits and Systems for Video Technology. 14, 1 (January 2004), 21-30. DOI=10.1109/TCSVT.2003.818350 http://dx.doi.org/10.1109/TCSVT.2003.818350

[28] Anil K. Jain, Arun Ross, Sharath Pankanti. "A Prototype Hand Geometry-based Verifcation System", In Proceedings of 2nd International Conference on Audio- and Video-based Biometric Person Authentication (AVBPA), Washington D.C., pp.166-171, March 22-24, 1999.

[29] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. "Face recognition: A literature survey". ACM Computing Surveys. 35, 4 (December 2003), 399-458. DOI=10.1145/954339.954342 http://doi.acm.org/10.1145/954339.954342

[30] Rick Joyce and Gopal Gupta. "Identity authentication based on keystroke latencies", Communications of the ACM ,33, 2 (February 1990), 168-176. DOI=10.1145/75577.75582 http://doi.acm.org/10.1145/75577.75582

[31] Davrondzhon Gafurov, Kirsi Helkala, Torkjel Søndrol. "Biometric Gait Authentication Using Accelerometer Sensor", Journal of Computers, Vol. 1, No. 7, October 2006.

[32] Roberto Brunelli and Daniele Falavigna. "Person Identification Using Multiple Cues", IEEE Transactions on Pattern Analysis and Machine Intelligence. 17, 10 (October 1995), 955-966. DOI=10.1109/34.464560 http://dx.doi.org/10.1109/34.464560

[33] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. "Touch me once and I know it's you!: implicit authentication based on touch screen patterns", In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12). ACM, New York, NY, USA, 987-996. DOI=10.1145/2207676.2208544 http://doi.acm.org/10.1145/2207676.2208544

[34] Ioannis Leftheriotis. "User authentication in a multi-touch surface: a chord password system" In CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13). ACM, New York, NY, USA, 1725-1730. DOI=10.1145/2468356.2468665 http://doi.acm.org/10.1145/2468356.2468665

[35] Ming Ki Chong, Gary Marsden, and Hans Gellersen. “GesturePIN: using discrete gestures for associating mobile devices”, In Proceedings of the 12th international conference on Human computer interaction with mobile devices and services (MobileHCI '10). ACM, New York, NY, USA, 261-264. DOI=10.1145/1851600.1851644 http://doi.acm.org/10.1145/1851600.1851644

[36] Android Developers, Uiautomator. https://developer.android.com/tools/help/uiautomator/index.html

[37] Volker Roth, Kai Richter, and Rene Freidinger. “A PIN-entry method resilient against shoulder surfing”, In Proceedings of the 11th ACM conference on Computer and communications security (CCS '04). ACM, New York, NY, USA, 236-245. DOI=10.1145/1030083.1030116 http://doi.acm.org/10.1145/1030083.1030116

[38] T. Perkovic, M. Cagalj, and N. Rakic. “SSSL: shoulder surfing safe login”, In Proceedings of the 17th international conference on Software, Telecommunications and Computer Networks (SoftCOM'09). IEEE Press, Piscataway, NJ, USA, 270-275.

[39] Alice Boit, Thomas Geimer, and Jorn Loviscach. “A random cursor matrix to hide graphical password input”, In SIGGRAPH '09: Posters (SIGGRAPH '09). ACM, New York, NY, USA, Article 41, 1 pages. DOI=10.1145/1599301.1599342 http://doi.acm.org/10.1145/1599301.1599342

[40] Rohit Ashok Khot, Ponnurangam Kumaraguru, and Kannan Srinathan. “WYSWYE: shoulder surfing defense for recognition based graphical passwords”, In Proceedings of the 24th Australian Computer-Human Interaction Conference (OzCHI '12), ACM, New York, NY, USA, 285-294. DOI=10.1145/2414536.2414584 http://doi.acm.org/10.1145/2414536.2414584

[41] Rachna Dhamija and Adrian Perrig. “Deja; Vu: a user study using images for authentication”, In Proceedings of the 9th conference on USENIX Security Symposium - Volume 9 (SSYM'00), Vol. 9. USENIX Association, Berkeley, CA, USA, 4-4.

[42] Mary Brown and Felicia R. Doswell. “Using passtones instead of passwords”, In Proceedings of the 48th Annual Southeast Regional Conference (ACM SE '10). ACM, New York, NY, USA, Article 82, 5 pages. DOI=10.1145/1900008.1900119 http://doi.acm.org/10.1145/1900008.1900119

[43] Andrea Bianchi, Ian Oakley, and Dong Soo Kwon. “The secure haptic keypad: a tactile password system”, In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10). ACM, New York, NY, USA, 1089-1092. DOI=10.1145/1753326.1753488 http://doi.acm.org/10.1145/1753326.1753488

[44] Alain Forget, Sonia Chiasson, and Robert Biddle. “Shoulder-surfing resistance with eye-gaze entry in cued-recall graphical passwords”, In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10). ACM, New York, NY, USA, 1107-1110. DOI=10.1145/1753326.1753491 http://doi.acm.org/10.1145/1753326.1753491

[45] Julie Thorpe, P. C. van Oorschot, and Anil Somayaji. “Pass-thoughts: authenticating with our minds”, In Proceedings of the 2005 workshop on New security paradigms (NSPW '05). ACM, New York, NY, USA, 45-56. DOI=10.1145/1146269.1146282 http://doi.acm.org/10.1145/1146269.1146282

[46] Nitesh Saxena and James H. Watt. “Authentication technologies for the blind or visually impaired”, In Proceedings of the 4th USENIX conference on Hot topics in security (HotSec'09). USENIX Association, Berkeley, CA, USA, 7-7.

[47] T. Dierks, E. Rescorla. "The Transport Layer Security (TLS) Protocol, Version 1.2", August 2008.

[48] Mason, Andrew G. “Cisco Secure Virtual Private Network”. Cisco Press, 2002, p. 7.

[49] Marc Shapiro. "Structure and Encapsulation in Distributed Systems: the Proxy Principle", In Proceedings of the 6th IEEE International Conference on Distributed Computing Systems (ICDCS), Cambridge MA (USA), May 1986.

[50] Roger Dingledine, Nick Mathewson, and Paul Syverson. “Tor: the second-generation onion router”, In Proceedings of the 13th conference on USENIX Security Symposium (SSYM'04), Vol. 13. 2004 USENIX Association, Berkeley, CA, USA, 21-21.

[51] Do Not Track. "Do Not Track - Universal Web Tracking Opt Out", http://donottrack.us. Retrieved 4/2014

[52] Adblock Plus. "Adblock Plus : About", https://adblockplus.org/en/about. Retrieved 4/2014

[53] Evidon, Inc. "About Ghostery", https://www.ghostery.com/en/about. Retrieved 4/2014

[54] Braden Kowitz and Lorrie Cranor. “Peripheral privacy notifications for wireless networks”, In Proceedings of the 2005 ACM workshop on Privacy in the electronic society (WPES '05). ACM, New York, NY, USA, 90-96. DOI=10.1145/1102199.1102217 http://doi.acm.org/10.1145/1102199.1102217

[55] Sunny Consolvo, Jaeyeon Jung, Ben Greenstein, Pauline Powledge, Gabriel Maganis, and Daniel Avrahami. “The Wi-Fi privacy ticker: improving awareness & control of personal information exposure on Wi-Fi”, In Proceedings of the 12th ACM international conference on Ubiquitous computing (Ubicomp '10). ACM, New York, NY, USA, 321-330. DOI=10.1145/1864349.1864398 http://doi.acm.org/10.1145/1864349.1864398.

[56] Simon Khalaf. “Apps Solidify Leadership Six Years into Mobile Revolution,” Flurry, http://www.flurry.com/bid/109749/Apps-Solidify-Leadership-Six-Years-into-the-Mobile-Revolution, 2014.

[57] Google, Inc. Google Glass. http://www.google.com/glass/start/

[58] Rahul Raguram, Andrew M. White, Dibyendusekhar Goswami, Fabian Monrose, and Jan-Michael Frahm. 2011. iSpy: automatic reconstruction of typed input from compromising reflections. In Proceedings of the 18th ACM conference on Computer and communications security (CCS '11). ACM, New York, NY, USA, 527-536. DOI=10.1145/2046707.2046769 http://doi.acm.org/10.1145/2046707.2046769

[59] Erika McCallister, Tim Grance, Karen Scarfone. Guide to Protecting the Confidentiality of Personally Identifiable Information (SP 800-122). National Institute of Standards and Technology, http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf

[60] Android Developers, Android Debug Bridge. https://developer.android.com/tools/help/adb.html

[61] Kino. The Kino Project. http://sourceforge.net/projects/pskino/

[62] Screen Concealer. http://screenconcealer.com/

[63] U.S. Census Bureau, Population Division, Population Analysis & Evaluation Staff. http://factfinder.census.gov/

[64] Jonathan Lampe. “Beyond Password Length and Complexity”, The Infosec Institute. January, 2014. http://resources.infosecinstitute.com/beyond-password-length-complexity/

[65] Oculus Rift - Virtual Reality Headset for 3D Gaming. Oculus VR, LLC. https://www.oculus.com/

[66] Samsung Gear VR. Samsung Electronics Co., Ltd. https://www.samsung.com/global/microsite/gearvr/

[67] 3M Privacy and Screen Protectors. The 3M Company. http://solutions.3m.com/wps/portal/3M/en_EU/3MScreens_EU/Home/PrivacyFilters/

[68] Lenovo Sun Visor. Lenovo Group Ltd. http://blog.lenovo.com/en/blog/see-in-the-light

[69] Becky Stern. Compubody Socks or Knitted Body-Technology Interfaces. Sternlab. http://sternlab.org/?p=90

[70] OpenSignal, Inc. “Android Fragmentation Visualized”, August, 2014. http://opensignal.com/reports/2014/android-fragmentation/

[71] Sonia Chiasson, P. C. van Oorschot, and Robert Biddle. "A usability study and critique of two password managers," In Proceedings of the 15th conference on USENIX Security Symposium (USENIX-SS'06), Vol. 15, 2006. USENIX Association, Berkeley, CA, USA, p. 1.