Cashtags: Prevent Leaking Sensitive Information through Screen Display

Michael Mitchell, An-I Wang
Department of Computer Science
Florida State University
Tallahassee, FL, USA
{mitchell, awang}@cs.fsu.edu

Peter Reiher
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA, USA
reiher@cs.ucla.edu

***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, to a significant degree, open. Shoulder surfing is becoming a viable threat in a world where users are frequently surrounded by high-power cameras and sensitive information from recorded images can be extracted with 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 access sensitive information in public without the fear of visual leaks. This is accomplished by intercepting sensitive data elements before they are displayed on screen, then replacing them with non-sensitive information. In addition, the system provides a means of computing with sensitive data in a non-observable way. All of this is accomplished while maintaining full functionality and legacy compatibility across applications.

***Index Terms*—Mobile privacy, shoulder surfing.**

I. Introduction

Shoulder surfing is becoming 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 via the screen display. Personal 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 emerging wearable computing devices such as Google Glass [57]. Additionally, the trend towards multicore machines, GPUs, and cloud computing makes computing cycles much more accessible and affordable for criminals or even seasoned hobbyists, seeking 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%.

A. *The shoulder surfing threat*

The threat of exposing sensitive information on screen to bystanders is real. In a recent visual data survey of IT professionals, 85% of those surveyed admitted there have been cases when they were able to see unauthorized sensitive on-screen data, 82% admitted that there have been cases where their own sensitive on-screen data could be viewed by unauthorized personnel, and 82% had little or no confidence that users in their organization would protect their screen from sensitive data exposure to unauthorized personnel [1]. These results are consistent with other surveys indicating that 76% of respondents were concerned about people observing their screens in public [2], while 80% admitted that they have attempted to shoulder surf the screen of a stranger in a public location [3].

The future projection of the shoulder-surfing threat is even worse, as mobile devices are replacing desktop computers. Mobile device sales now account for over 73% of annual technical device purchases [4]. Employees more frequently take their work with them on the go; by 2015, the world's mobile worker population will reach 1.3 billion [5]. This is highest in the US, where more than 80% of the workforce continues working when they have left the office [6], and figures suggest that 67% of employees regularly access sensitive data outside at places where they cannot do so safely [2]. While some organizations have implemented specific guidelines and practices to reduce this risk, 44% do not have any defined 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].

B. *The dangers are everywhere*

Visual exposure of sensitive information in the form of observation-based attacks can come in many forms. Mobile

devices with cameras are nearly ubiquitous. There now exist more than 3 billion digital camera phones in circulation [4]. These devices are evolving rapidly, with newer models capable of capturing images at over 40 megapixels of resolution and over 10 times optical zoom for under $100 [7]. Visual exposure can also be captured by one of the billions of security devices in existence. These high-resolution and often insecure cameras are everywhere, especially in major metropolitan areas. For example, figures suggest the average resident of London is captured on CCTV over 300 times every day [9]. Finally, but no less threateningly, sensitive data can be exposed by simple human sight.

Observation-based attacks can also be much more complex. Increasingly sophisticated tools and systems have been developed to capture and exploit sensitive user data. Partial images can be merged, sharpened, and reconstructed, even from reflections. Optical Character Recognition (OCR) is becoming much more capable, with over 40 years of innovation. Offline and cloud-based OCR solutions are highly accurate with only a small percentage of error in recognition. Embedded OCR solutions are inexpensive and capable even on low-end hardware devices [10].

Personal information exposure can also make other attacks possible. The capture of just a small number of personal information elements can greatly increase the risk of other threats including social engineering attacks, phishing, and other personal identity theft threats.

C. The consequences can be severe

Observation-based information leaks can lead to significant personal and business losses. Recently, an S&P 500 company's profit forecasts were leaked as a result of visual data exposure. The vice president was working on the figures on a flight while sitting next to a journalist [4]. In a different case, British government documents were leaked when a senior officer fell asleep on a train, thereby permitting another passenger to photograph sensitive data on his screen [11]. In another case, security cameras captured the private details of Bank of America clients through the bank's windows [12]. In yet another case, sensitive personal information relating to the United Kingdom's Prince William was captured and published as a result of on-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 they felt that no one would be able to see their screen [2].

D. Current solutions

Several techniques have been developed to limit the visual exposure of sensitive private information. However, the primary focus of these systems has been limited to preventing the visual leakage of password entries [22, 23, 24, 25, 33, 34, 35]. Once the user has been successfully authenticated, all 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, a situation that motivates our work.

II. CASHTAGS

We present Cashtags[1]: a system that defends against observation-based attacks. The system allows a user to access sensitive information in public without the fear of visual privacy leaks.

A. 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.

B. User model

Conceptually, Cashtags is configured with a user-defined list of sensitive data items, each with a respective Cashtags alias or a cashtag (e.g., $visa to represent a 16-digit credit-card number; see other examples in Table II). Then, whenever the sensitive term would be displayed on screen, the system displays the pre-defined alias instead (Fig 2.1). At the point at which the sensitive data would be used internally by the device or an app, cashtags will be replaced by the sensitive data items represented by the alias, allowing whatever login, communication, transmission, or upload to proceed normally.

Also, a user can directly type in a cashtag in place of the sensitive term, permitting more complex data-sensitive tasks such as filling out an application for a credit card or loan without risk of observation from a bystander. In addition, cashtags are easier to remember than the actual information itself. For example, $visa can be used as a shortcut for entering a 16-digit credit card number.

C. Design overview

Although conceptually simple, the design of 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, at their common textual rendering library routines; for users, at routines to handle software keyboards as well as 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, cut and paste, etc. Thus, users

[1] Cashtag, an amalgam of the words *cash* and *hashtag*, serving as an easy-to-remember alias for a valuable sensitive personal identifier. A cashtag alias consists of a dollar sign followed by an arbitrary string of printable characters.

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 for a name).

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 centralized, password-protected repository.

TABLE II
SAMPLE MAPPING OF SENSITIVE DATA TO CASHTAG ALIASES

Type	Actual	Alias
Name	John Smith	$name
Email	jsmith@gmail.com	$email
Username	Jsmith1	$user
Password	p@ssw0rd	$pass
Street Address	123 Main St.	$address
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

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

III. CASHTAG DESIGN

This section will present the design options for each Cashtags design point and explain how we arrived at the current design.

A. *Intercepting 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 various display data paths under the Android application development platform. Although iOS and Windows use different platforms, the display data paths generally can be mapped from one platform to another.

A typical app displays information on screen 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 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 technologies are computationally cheap enough to be used for launching shoulder surfing attacks, they are still too heavyweight for deployment in the display data path, which is critical for user interactions. Also, replacing sensitive bitmaps with non-sensitive ones would pose other obstacles we would like to avoid.

Another extreme is to intercept at the app level, where the sensitive information is introduced. Potentially, we can modify a few top apps and capture a majority of cases where sensitive information is used. For instance, custom e-mail applications or browsers could offer protection for task-specific usages. However, such solutions may restrict users to using a specific tool for a specific task. In addition, statistics show that specific app usage accounts for 86% of user time, trending away from general-purpose browsers [56]. Thus, the burden of incorporating our features could spread to a much wider range of app developers, which is undesirable. Further, new apps and updates to old apps would not automatically include the desired protection.

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

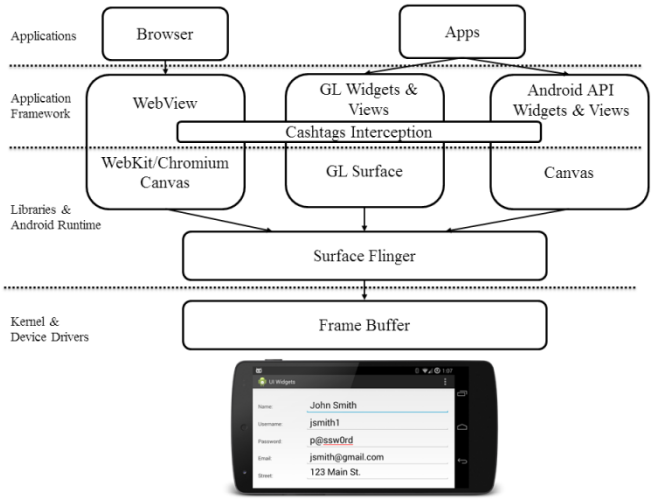

Fig. 3.1. Display data paths for the Android platform.

B. User interface

Early design: In our early user-interface design, a user defines 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 turned out to be problematic in a number of ways.

One way to achieve this effect is to add a separate software keyboard that a user would use whenever they want to input sensitive information. Essentially, this keyboard would be an app with elevated privilege to offer input across applications, and it would be easier to port across platforms, deploy, install, and update. However, changing keyboards amidst a stream of input is cumbersome in practice. This method would result in the loss of functionality offered in default keyboards, including swipe-based inputs, emoticon support, auto correction, and custom dictionaries.

One step further is to replace the default keyboard with ours, which provides ways (e.g., a screen tapping sequence) to switch modes between normal entries with sensitive entries. By doing so, we can retain legacy functionalities such as auto correction. The user learning curve would be less steep, since no keyboard switching would be involved. On the other hand, the development effort of this approach would be significantly higher, and it would be harder for novice users to install our software, namely, by carrying out the replacement of the default keyboard.

Direct input of cashtags: While there are other input interface options than these, the need to perform cumbersome switches of input modes so that the aliases can appear as normal text seems superfluous in many contexts (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 explicitly 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 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 alternative (albeit with some potential for information leak) is for a user to attempt to enter the initial characters of a sensitive data item. As soon as the auto completion detects that Jo, for example, is likely to mean Joe, it will be automatically masked with the cashtag $John. The user then can choose $John and proceed.

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

C. Accessing sensitive information

One design issue involves when to convert cashtags back to the sensitive data for accesses by apps. Normally, when an app wants to access the sensitive information and sends it back to the hosting server, we need to make sure that the conversion is performed prior to access, so that the app would not be able to cache, store, or transmit the cashtags. The main concern is that cashtags may not adhere to the type or formatting constraints and break an app inadvertently.

Another thing we need to make sure of is 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 also certain exceptions where it is desirable to directly operate 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.

D. 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; Smith, John Q). Accounts and social security numbers can be represented using different spacing and/or hyphenation schemes (e.g., 123456789; 123-45-6789; 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) in place of cashtags to represent certain types of sensitive information (e.g., social security numbers).

E. Deployment and development models

To avoid modifying individual applications, we considered two options to provide system-level changes: (1) custom system firmware images (ROMs) or (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 for modification.) 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 explicitly 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 would break vendor over-the-air update functionality.

Instead, we used a code-injection framework, which dynamically introduces overriding routines as a library, incorporated into execution prior to the starting of apps. Code

injection offers more streamlined development, as standard user application development tools can be used. In addition, these modules can be more easily deployed since they can be distributed as applications. Because code injection only relies on the underlying system using the same set library, the deployment is much more portable and less coupled with the exact versions and configurations of system firmware.

F. Cashtags App and Repository

Cashtags aliases and sensitive data items are maintained in an internal 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 $first_name to Joe. If Cashtags is enabled, the screen will show that $first_name is mapped to $first_name. When the entry is saved, Joe will be mapped to Joe. Thus, Cashtags is always excluded from servicing itself. Individual application packages can be excluded for lack of relevance to sensitive information exposure or for performance issues (e.g. games, application launchers, home screens).

IV. 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 display data path in the Android context, then explain the code-injection framework, and finally discuss the details of how various display data paths are intercepted and how cashtags are stored.

A. Android display elements

Fig 3.1 has already shown a top-level view of various ways Android apps and browsers display information on the screen. This section provides further background on Android terminologies before we begin detailing our implementation. The corresponding terminologies for iOS and Windows are listed in Table IV.

Android on-screen display is composed of views, layouts, and widgets. View is the base class for all on screen user interface components. All visual elements are descendants of this base class.

TABLE IV
WIDGET TERMINOLOGY ON OS PLATFORMS

	Android	Apple	Windows
Text Labels	TextView	UITextView	TextBlock
OpenGL Text	GLES20Canvas	GLKView	Direct3D
Editable Text	TextView	UITextView	TextBlock
Webapp Text	WebView	UIWebView	WebView
Browser/Web Views	WebView	UIWebView	WebView

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 more complex custom composites to be defined.

Collectively, this tree of layouts and widgets is called the view hierarchy. When the screen canvas is drawn, the view hierarchy is converted from logical interface components into a raw 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 screen through several mechanisms (Fig 3.1), the most common being through the TextView widget. Fonts, styles, colors, and so forth can be applied to specify how these are displayed. An EditText is an extension of the TextView that provides an interface for text input. This input can come from the user via the on-screen software keyboard, (integrated, plugged, or wirelessly connected) hardware keypads, voice input, gestures, and so forth. Like TextView, these widgets can be pre-filled with text by the app internally. They can also be set through suggestion or auto-correction interfaces.

Text can also be rendered on screen via OpenGL Canvas or other graphic rendering libraries. Unlike the EditText, this class does not inherit from the base TextView, although similar interfaces do exist.

Text can further be rendered on-screen from HTML and Javascript via browser rendering engines such as WebKit or Chromium. This includes mobile web browsing applications as well as many other cross-platform web app APIs such as Phonegap, Apache Cordova, and JQuery Mobile.

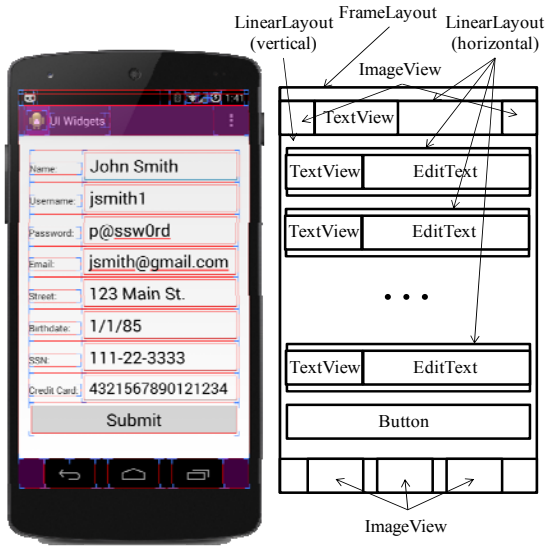

Fig. 4.1 Decomposition of on screen views, layouts, and widgets of a simple app input forms.

B. *Android code-injection framework*

Cashtags uses the Android Xposed code-injection framework to intercept and modify the behavior of text widgets at runtime, without being tied to specific system firmware images. The development cycle is also 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 additional 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 a base method calls, following the completion of the base method call, or in place of the base method. 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. Fig. 4.2 shows the API provided by Xposed for injecting method, constructor, and fields.

```
hookAllMethods() / hookAllConstructors()
findMethod() / findConstructor() / findField()
callMethod() / callStaticMethod() / newInstane()
getXXXField() / setXXXField()
getStaticXXXField() /setStaticXXXField()
getAdditionalXXXField() / setAdditionalXXXField()
```

Fig. 4.2. Code injection API provided by XposedBridge. XXX denotes the specified data type, boolean, int, float, etc.

C. *Sensitive data intercepting points*

With the background of 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. 4.3), we hooked TextView (android.widget.TextView) to intercept static sensitive text. For input, we hooked EditText (android.widget.EditText) to capture sensitive data or cashtags entered via on-screen software keyboard, (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).

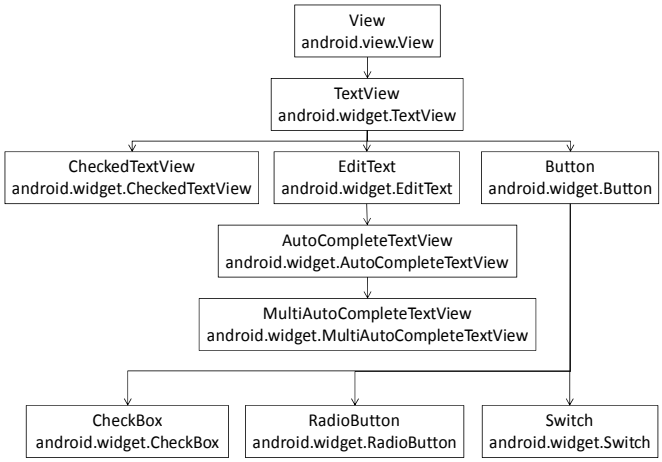

Fig. 4.3. Simplified Android screen widget view hierarchy.

D. *TextView*

Fig. 4.4 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 TextView 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.

Fig. 4.5 and Fig. 4.6 show how Cashtags interacts with TextView and EditText objects. When these getText() and setText() methods are called by the app or through system processes like auto correct or to be rendered on screen, Cashtags will determine whether to return the alias or the sensitive data, depending on the caller.

```
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. 4.4. 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.

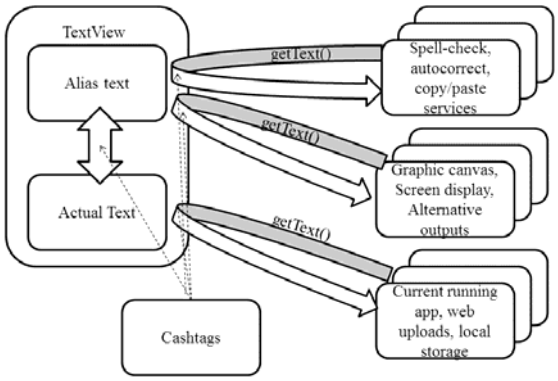

Fig. 4.5. Interactions among Cashtags, TextView, and other software components. getText() returns cashtag or actual text depending upon the service making the request.

E. EditText

EditText objects are more complex since additional actions can be performed by the user, app, or system to modify on-screen text. 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 as they are also hierarchically descendant from the base TextView.

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.

This functionality is also achieved through the view OnFocusChangeListener (android.view.View .OnFocusChangeListener). This event handler works at the granularity of the full text field rather than individual character 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 the cursor remains in that text field. Input of 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 respective OnFocusChangeListener or TextWatcher is attached. User settings allow activation of either or both options within the app settings.

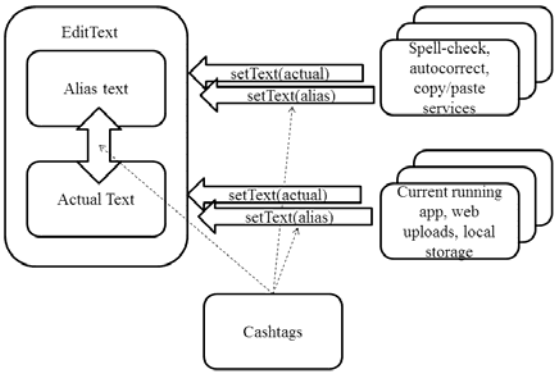

Fig. 4.6. Interactions among Cashtags, EditText, and other software components. setText() returns cashtag or actual text depending upon the service making the request.

F. OpenGL Canvas

The implementation solution for OpenGL ES Canvas is quite similar in simplified form 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 additional manipulation is necessary beyond drawText().

G. WebView

Distinct from the previous screen widgets, rendering occurs independently of the native UI data path via underlying WebKit or Chromium browser engines. The relevant interception points for screen rendering for these are all below the accessible Android/Java layer and are not able to be code-injected though the same mechanisms used for previous screen widget cases. Using custom compilations of the browser engines with similar widget display interception was explored, but abandoned 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.

H. Cashtags Repository

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

V. Evaluation Methods

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

A. API coverage evaluation

The first test is for Android API coverage. We focus on the TextView and EditText display data paths, which account for more than 86% of usage hours for mobile devices [56]. The selected sensitive information (Table II) is based on the Personally Identifiable Information (PII) chosen categorically based on US government and NIST standards [59]. We enumerate all combinations of input phrase type (e.g., numbers, strings, etc.), case sensitivity, common widget, layout, theme, and other configuration options for these data paths. Each combination is used to demonstrate that the PII terms are not displayed on screen from the app internally, as user input of the sensitive data directly, or as user input of cashtag alias. In all three cases, we also demonstrate that the PII term is correctly returned from Cashtags when used internally by the app.

This totals 1,728 tests for static text widgets and inputs, with 526 additional test cases for widgets that permit user input via both software keyboards as well as physical devices (on-board hardware, USB or wireless input devices). The full list of configurations is shown in Table V.I.

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

For each combination, we also demonstrate that both text input as an actual sensitive term and cashtag are correctly converted to the actual sensitive term when accessed internally 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 to Google Sheets/Form. The uploaded actual 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 actual and cashtag form, prevented the display on screen of the sensitive term, and determined correctly when to convert back to the sensitive data.

TABLE V.I
Android API Test Combinations

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

B. App coverage evaluation

The Google Play market has millions of published applications accessible by thousands of different hardware devices, making the enumeration of all possible users, devices, and application scenarios infeasible. Thus, we chose a representative subset of popular apps to demonstrate app coverage of Cashtags. Categorically, these application types are email, messaging, social media, cloud and local storage, office, and finance. Table V.II 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.

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

TABLE V.II
PER-CATEGORY APP TEST TASKS

Email: AOSP Email, Gmail, K9 Mail:

A user reads an email containing a sensitive term and its corresponding cashtag. A Cashtags-enabled system should display the email with two instances of the cashtag.

A user composes an email with a sensitive term and its cashtag. A remote system not running Cashtags should display the email with two instances of the sensitive term.

Messaging: Messaging, Google Hangouts, Snapchat:

A user reads a message containing a sensitive term and its cashtag. A Cashtags-enabled system should display a message containing two instances of the cashtag.

A user composes a message with a sensitive term and its cashtag. A remote system not running Cashtags should receive the message containing two instances of the sensitive term.

Social: Facebook, Twitter, Google+:

A user reads text containing a sensitive term and its cashtag from tweet/post/update. A Cashtags-enabled system should display the tweet/post/update containing two instances of the cashtag.

A 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 instances of the sensitive term.

Storage: Dropbox, MS OneDrive, File Manager:

A 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.

A user creates a file with a sensitive term and its cashtag. A remote system not running Cashtags should see the file containing two instances of the sensitive term.

Office: GoogleDocs, MS Office Mobile, QuickOffice:

A user reads a document containing a sensitive term and its cashtag. A Cashtags-enabled system should display the document with two instances of the cashtag.

A user creates a document containing a sensitive term and its cashtag. A remote system not running Cashtags should see two instances of the sensitive term.

Finance: Google Wallet, Paypal, Square:

A user reads a document containing a sensitive term and its cashtag. A Cashtag-enabled system should display the document with two instances of the cashtag.

A user creates a document containing a sensitive term and its cashtag. A remote system not running Cashtag should see two instances of the sensitive term.

TABLE V.III
APP COVERAGE EVALUATION

	User Input Actual	User Input cashtag	Remote Success Actual	Remote Success cashtag
Email				
AOSP Email	√	√	√	√
Gmail	√	√	√	√
K9 Mail	√	√	√	√
Messaging				
Messaging	√	√	√	√
Google Hangouts	√	√	√	√
Snapchat	√	√	√	√
Social				
Facebook	√	√	√	√
Twitter	√	√	√	√
Google+	√	√	√	√
Storage				
Dropbox	√	√	√	√
MS OneDrive	√	√	√	√
File Manager	√	√	√	√
Office				
Google Docs	√	√	√	√
MS Office Mobile	√	√		
QuickOffice	√	√	√	√
Finance				
Google Wallet	√	√	√	√
Paypal	√	√	√	√
Square	√	√	√	√

Table V.III shows that Cashtags using test cases from market apps shows correct behavior for 97% of task and app combinations, except the MS Office Mobile tests. The reason these tests does not work is due to the custom View used for the primary user interaction. This View (id: docRECanvasHost) is not a descendant of an EditText so is not intercepted by Cashtags. All other apps tested have user input through an EditText, or a custom class inheriting from an EditText. Custom views beyond this scope could be made to work with Cashtags using case-specific handling for the internal functions and parameters that map to the equivalent EditText function.

C. Overhead

In terms of overhead, Cashtags was evaluated based on the incremental lag on the system. To perform this test, a modified version of the Android API coverage test (Section A) was run with and without Cashtags enabled. Screenshots, layout hierarchy dumping, and all other non-essential automation elements were removed prior to test execution. Test execution durations are compared, and additional incremental lag introduced by the system is calculated. This test is run with and without the remote data verification to determine the effects of network lags on our system overhead.

Fig. 5.1 show the Cashtags system incurs an average 1.9% increase in test execution duration. For tests including remote verification, Cashtags incurred an average of a 1.1% increase over baseline tests. For tests excluding the time consuming remote verification, Fig. 5.2 shows that Cashtags incurred an average of 2.6% over baseline. Therefore, under such conditions, the additional overhead of Cashtags would not be perceivable to the user.

Testing was also repeated using more cashtag entries, with 50 and 100 items, which is significantly higher than the list of terms specified by PII. Fig. 5.3 and Fig. 5.4 show the results of these test runs for both system and user input data, using tests with and without the task inclusion of a web request. Due to the current data structure, 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 reboot to take full effect. While this operation is not in the common critical path, the additional overhead for this operation is relevant. The results of the boot lag are shown in Fig. 5.5.

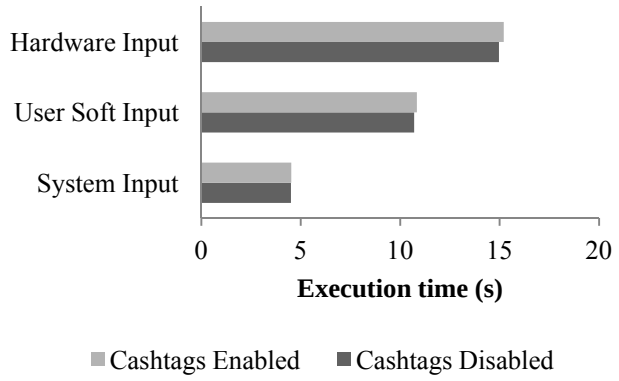

Fig. 5.1. Comparison of mean app task execution time with and without Cashtags enabled, using system, software and hardware text input with web request for tests. Hardware input refers to input from physically or wirelessly connected hardware keyboard and Software Input to input from on screen software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

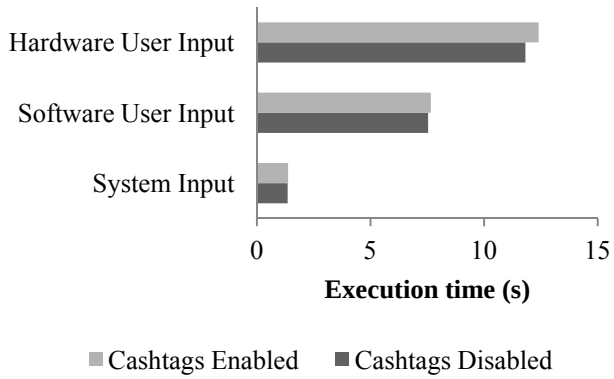

Fig. 5.2. Comparison of mean app task execution time with and without Cashtags enabled, using system, software and hardware text input without web request for tests. Hardware input refers to input from physically or wirelessly connected hardware keyboard and Software Input to input from on screen software keyboard.

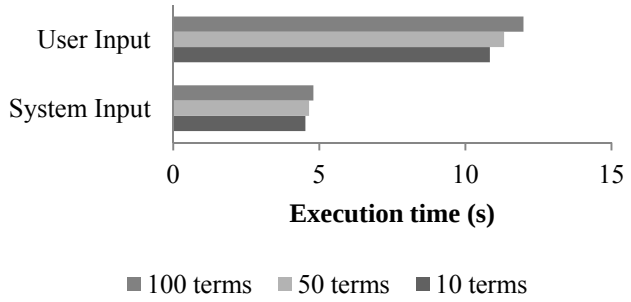

Fig. 5.3. Comparison of mean app task execution time with an increasing number of cashtag entries, using system and user inputs with web request for tests.

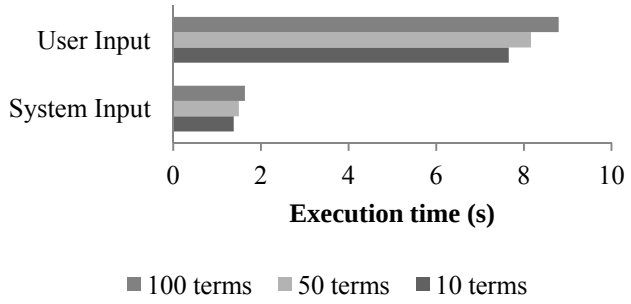

Fig. 5.4. Comparison of mean app task execution time with an increasing number of cashtag entries, using system and user inputs without web request for tests.

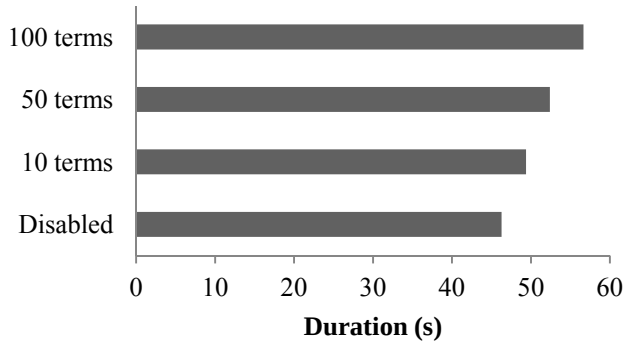

Fig. 5.5. Comparison of device startup times with a varying number of cashtag entries and with system disabled.

VI. Related Work

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

A. Visual authentication protection

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

1) Password managers

Perhaps the most common method of securing against an observation-based attack is the use of password managers. These managers are software tools that 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 applications without the need to remember each of them individually.

2) Hardware-based authentication

Other related work involves external physical devices to supplement or replace the need to enter passwords. These techniques utilize specialized USB dongles [17], audio jacks [18], short-range wireless communication using NFC [19], or Bluetooth connections [20] to connect to the authenticating machine.

3) Graphical passwords

Another technique to help guard against information leaks from visual attacks is the use of graphical passwords or Graphical User Authentication (GUA) [22]. Such techniques remove the alpha-numeric password from the equation and replace it with the use of series of images, shapes, and colors. Common techniques present the user with a series of human faces that must be clicked on in sequence [23], object sequences as part of a story [24], or specific regions within a given image that must be clicked in sequence [25].

4) Biometrics

Biometric authentication mechanisms can be generalized as changing or augmenting password entry (something one knows), with a feature unique to one's personal biology (something one is). There are many inherent physiological characteristics that are sufficiently unique to identify and differentiate one individual from another. 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, in contrast to biometric identifiers, including keystroke latency [30], gait [31], and voice [32] can also be used for authentication purposes.

5) Gesture-based authentication

Closely related to both GUA techniques and biometric solutions are gesture-based authentication techniques. These methods allow the user to perform specific tap [33], multi-finger presses [34], or swipe sequences on-screen [35] to represent a password.

6) Cognitive challenges, Obfuscation and confusion

Other techniques have attempted to make games of the authentication procedure [37]. Instead of a single password or phrase, these techniques utilize challenge-response questions and use of cognitive tasks to increase the difficulty of the login session [38]. Other techniques have attempted to remedy the shortcomings of password-based authentication through obfuscation and confusion to a visual observer. They utilize the hiding of cursors [39], confusion matrices [40], and recognition [41] rather than recall-based methods, to trick and confuse onlookers.

7) Alternate sensory inputs

Additional work has been done utilizing other biological sensory inputs to replace or augment password-based authentication. These systems can address two separate parts of the authentication process; the cue to the input, or the actual input itself.

In the first case, the additional sensory input serves as a non-observable instruction or hint to the required passphrase entry. These 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 phrase corresponding to the cue using traditional input methods.

In the second case, the auxiliary sense serves as the input mechanism itself. These systems extend GUAs by requiring sequential graphical inputs but use mechanics like 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].

B. Digital Communication Channel Protection

Many protocols and systems have also 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 location, and anonymization of IP addresses.

Many other solutions have been developed to enhance security and privacy at the browser level. Do-not-track requests can be included in 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 advertising [52] as well as analytics, beacons, and other tracking mechanisms [53]. Other systems alert the user when specific privacy elements are leaked [54, prevent the transmission of sensitive data without explicit user permission [55], and cryptography secure access to sensitive data outside of trusted situations [16].

C. 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. All existing tools developed for encryption of data are not originally designed for such purposes.

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

VII. Discussion & Limitations

A. Coverage Limitation

Cashtags widget-level text manipulation works for apps that use standard text rendering methods. However, should developers deviate from such standards and create display data paths that do not inherit from the base text widgets, 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.

B. Common Name Issue

Commonly occurring names can result in certain side effects. 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. Now, John opens his mobile browser and googles for John Addams, John Travolta, or John Williams. All returned search results would be displayed with on-screen representations as $fname Addams, $fname Travolta, or $fname Williams, respectively.

While this may or may not be acceptable to the user, it could also have the unintended consequence of inadvertently visually leaking private data. If an on-looker was able to observe the above search queries in the situation above, and was aware of the operation of Cashtags, they might be able to derive the sensitive data from context; in this case, determining that the user making the searches is named John. This limitation is isolated to common phrases; most instances of numerical phrases would not be relevant to this issue.

C. Data Formatting

Data formatting and types is another issue. Many cases are handled through simple transformations of text fields, including the removal of spaces and symbols, and capitalization mismatches. However, on-screen data that expands between individual TextViews is not recognized, e.g., input fields for a credit card split into parts rather than combined into a single field. This could be handled by Cashtags if each part of the credit card number were individually added to the repository.

VIII. Future Work

In its current form, Cashtags is designed to protect against privacy leaks for the device owner. However, modification could be made to provide more generalized protection from on-screen data leaks, especially for business use cases. Many professions regularly access lists of data containing sensitive data elements. This use case is becoming more commonplace, as progressively more computing is being performed on mobile devices. Additional processing of text elements for specific patterns of text and other data could be applied to contextually determine which data fields may contain sensitive data. These fields could then be masked accordingly.

Other future work could improve the scalability of the sensitive data repository. The current implementation is optimized for coverage rather than performance. Disabling of specific classes of widgets unlikely to contain sensitive data is one solution. In addition, more efficient text processing methods and data structures can be considered.

Other future work could include the remote synchronization of Cashtags. Updates to sensitive actual and alias lists could be propagated to other devices automatically. Cashtags could also be modified to provide shared access for multiple users. Permissions could allow a user to share a cashtag for use by another without disclosing the sensitive data. In addition, this method would provide improved redaction of access to shared sensitive resource.

IX. 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 personal information. The evaluation of the system shows that this is accomplished efficiently, with minimal perceived overhead. The app coverage test confirms that the system is general purpose and maintains full functionality with nearly all tested common use cases. These results suggest that Cashtags will likely also work on most other mobile apps, providing unified, device-wide protection against shoulder surfing.

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