EyeDecrypt — Private Interactions in Plain Sight

Andrea G. Forte
AT&T Labs
Email: forte@att.com

Juan Garay*
Yahoo Research
Email: garay@yahoo-inc.com

Trevor Jim
AT&T Labs
Email: trevor@att.com

Yevgeniy Vahlis*
Bionym
Email: evahlis@gmail.com

Abstract

We introduce *EyeDecrypt*, a novel technology for privacy-preserving human-computer interaction. *EyeDecrypt* allows only authorized users to decipher data shown on a display, such as an electronic screen or printed material; in the former case, the authorized user can then interact with the system (*e.g.*, by pressing buttons on the screen), without revealing the details of the interaction to others who may be watching or to the system itself.

The user views the decrypted data on a closely-held personal device, such as a pair of smart glasses with a camera and heads-up display, or a smartphone. The data is displayed as an image overlay on the personal device, which we assume cannot be viewed by the adversary. The overlay is a form of augmented reality that not only allows the user to view the protected data, but also allows to the user to securely enter input into the system by randomizing the input interface.

EyeDecrypt consists of three main components: a *visualizable encryption* scheme; a dataglyph-based visual encoding scheme for the ciphertexts generated by the encryption scheme; and a randomized input and augmented reality scheme that protects user inputs without harming usability. We describe all aspects of *EyeDecrypt*, from security definitions, constructions and formal analysis, to implementation details of a prototype developed on a smartphone.

I. Introduction

Nowadays personal and sensitive information can be accessed at any time, anywhere, thanks to the widespread adoption of smartphones and other wireless technologies such as LTE and IEEE 802.11 (i.e., WiFi). This always-connected paradigm, however, comes at the expense of reduced privacy. Users access sensitive information on the train, on the subway and in coffee shops, and use public computers in airports, libraries and other Internet access points. Sensitive information is at the mercy of anyone in the user's proximity and of any piece of malware running on trusted and untrusted devices such as a personal laptop or a computer in a library. This applies not just to the content displayed on a monitor but also to the interaction users have with the system (e.g., typing a password or a social security number). Someone looking at the keyboard as one types in a password is as bad as showing the password in clear text on a login page.

We introduce *EyeDecrypt*, a technology aimed at protecting content displayed to the user as well as interactions the user has with the system (e.g., by typing). In particular, we do not trust the user's environment (e.g., "shoulder surfing"), nor do we trust the device the user interacts with as it may have been compromised (e.g., keyloggers).

As shown in Figure 1, in *EyeDecrypt* the content provider encrypts and authenticates content using *EyeDecrypt* and sends it to the device the user requested the content from (e.g., laptop, cellphone, ATM). Because the content arrives already encrypted to this untrusted device, any piece of malware running on it would not be able to learn anything meaningful about the content being displayed to the user. Similarly, the user interacts with the untrusted device using *EyeDecrypt* so that only the remote content provider learns the actual inputs provided by the user during the interaction (e.g., typed password or PIN code). A piece of malware such as a keylogger running on the untrusted device would not be able to learn what the user has typed.

Let us now provide some intuition on how *EyeDecrypt* works at a high level. If we print a document with extremely small fonts, this will appear as a collection of dots with no meaning. If, however, we take a very powerful magnifying lens, we will be able to read the part of the document right underneath the lens; further, by moving the lens around, we will be able to read the whole document. Anyone without the magnifying lens (i.e., a shoulder-surfer) will see just dots. *EyeDecrypt* provides a similar experience.

*Work done while at AT&T Labs

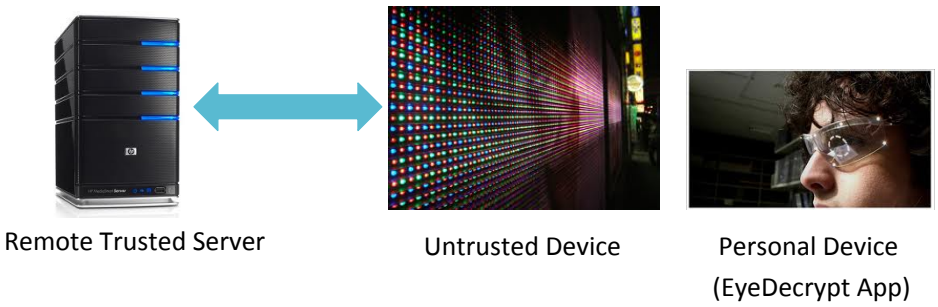


Fig. 1: *Basic setup*

In *EyeDecrypt* content is encrypted and visually encoded so that it appears as some pattern of dots, lines or other shape to anyone looking at it. In order to be able to decrypt such document or parts of it, users will have to use the *EyeDecrypt* app on their personal device (e.g., smartphone, Google Glass). Such app enables users to use the camera on their personal device as the "magnifying lens" described earlier. By leveraging the smartphone camera, for example, the *EyeDecrypt* app captures a part of the encrypted content, decrypts it and overlays the decrypted content on top of the camera view on the personal device—a form of augmented reality. By moving the smartphone around over the document, users will capture, decrypt and display different parts of the document. One key difference with the magnifying lens example is that the *EyeDecrypt* app will be able to decrypt a document only if it has the correct cryptographic keys for that document or that content provider. Just the *EyeDecrypt* app by itself is not enough to decrypt content.

Importantly, *EyeDecrypt* also protects users' interactions with the system. For example, in the case of a keyboard, a randomized keyboard layout can be encrypted and displayed to the user together with other encrypted content. The *EyeDecrypt* app will decrypt all content including this randomized keyboard layout and will superimpose such layout on the camera view as an overlay on the actual physical keyboard (i.e., using augmented reality). In doing so, there is now a random mapping between keys of the physical keyboard and keys of the randomized layout that the user can see. Any onlooker would see the user pressing,say, the 'A' key on the physical keyboard without knowing to which value it would actually map to in the randomized layout. In particular, the random mapping between physical keyboard and virtual keyboard would be known only to the user, the *EyeDecrypt* app and to the remote server that encrypted the content (see Figure 1). The untrusted device on which the encrypted document is displayed would not be aware of such mapping. Because of this, even a keylogger running on the untrusted device the user is interacting with, would not be able to learn the actual key values inputted by the user.

As mentioned above, *EyeDecrypt* aims at protecting against attacks on content displayed to the user as well as on information sent by the user (e.g., by typing). Such attacks may be due to shoulder surfing as well as to malware. While *EyeDecrypt* can leverage any device equipped with a camera, the type of device is important as different types of devices make *EyeDecrypt* more or less effective depending on the threat scenario. Let us look at a few settings.

In the most general case of shoulder surfing, the attacker can be anywhere in the victim's surroundings. In such a case, displaying decrypted content on a device such as a smartphone does not completely remove the possibility of someone being able to glance at the smaller screen of the phone even though the smaller screen of the phone does make it harder. A better solution in this scenario would be to use *EyeDecrypt* with a device such as Google Glass where the screen is very small and close to the eyes of the user making a shoulder-surfing attack much harder.

In a different type of shoulder surfing attack, the attacker has installed a small fixed hidden camera in close proximity of an ATM keypad so as to film the hands of users as they enter their PIN code. In such a scenario, a solution based on using *EyeDecrypt* with a smartphone would be perfectly fine as the hidden camera would not be able to capture the screen of the smartphone[1]. Similarly, using *EyeDecrypt* with a smartphone would be perfectly suitable to protect against an attack involving malware such as a keylogger.

When thinking about a shoulder-surfing attack it is natural to ask, "Why not just display content in a head-mounted display without any encryption?" The answer is that this would prevent shoulder surfing, but it assumes

[1]Naturally, we assume the user to be security conscious so as not to position the phone too close to the ATM keypad.

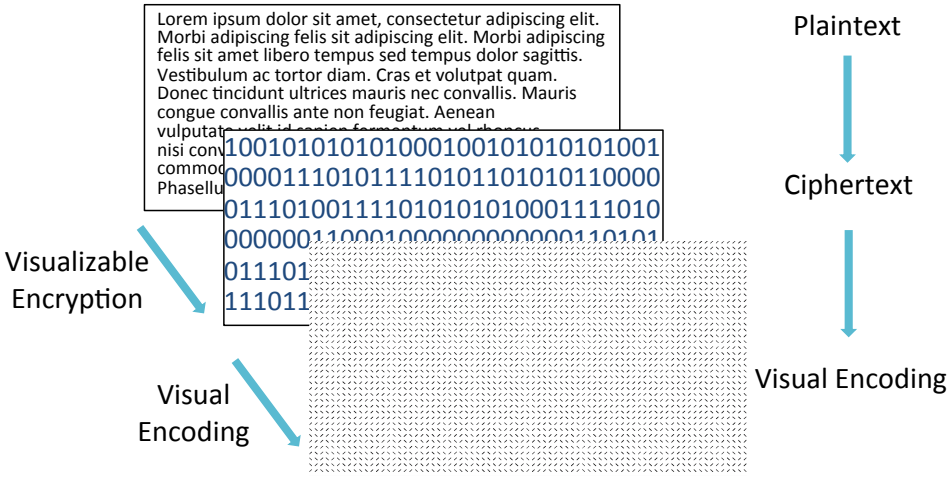


Fig. 2: *EyeDecrypt overview*

that the device that the head-mounted display is plugged into can be trusted. It does not help in the more difficult case of a modified ATM or compromised public terminal.

Another idea is to encrypt content at the server, send it to the untrusted device, and have the untrusted device forward it to the user's trusted personal device via wireless transmission, instead of using *EyeDecrypt*'s visual channel. This is definitely a viable solution and has its advantages, such as higher bandwidth. However, it has two significant downsides.

First, both the personal device and the untrusted device need to be equipped with the same wireless technology. It would not work, for example, with existing ATMs, which do not employ Bluetooth or WiFi. In contrast, *EyeDecrypt* works on any personal device equipped with a camera, and it makes no assumptions about the connectivity of the untrusted device; *EyeDecrypt* can work with existing ATMs, without requiring a hardware upgrade.

Second, wireless communication requires *secure pairing*. As in the ATM case, it may be that the user has never interacted with the public device before. Pairing is required to be sure that the personal device is communicating with the intended public device, and not some other device in the vicinity. Secure pairing by itself is a hard problem and one of the devices being untrusted (i.e., possibly misbehaving) makes it much harder. *EyeDecrypt* does not require pairing—the user knows where she is pointing her camera—making the whole process much more secure and user-friendly. Notably, most of the secure pairing solutions that have been proposed involve use of the visual channel, for exactly this reason. In other words, use of the wireless channel in these scenarios already requires some mechanism like *EyeDecrypt*. We discuss secure pairing in Section VI.

Lastly, *EyeDecrypt* also works with printed content such as passports, bank statements, medical records and any other type of sensitive material. If the human eye can see it, *EyeDecrypt* can protect it.

EyeDecrypt consists of three main components: an encryption mechanism which we term *visualizable encryption*; a suitable *visual data encoding* scheme; and a combination of augmented reality and randomization for secure input. Figure 2 shows how content is encrypted and then visually encoded.

Visualizable encryption is distinct from ordinary encryption in that information is captured incrementally, frame-by-frame, and even block-by-block within a frame, in a pan-and-zoom fashion. In addition, our notion of security refers to the security of the "*EyeDecrypt* activity" and not just to the security of the encryption scheme. As such, it must also take into account what the adversary is able to observe—not only ciphertext, but also the user's interaction (e.g., gesticulation) with the system. We believe that formally defining the security of these new applications is an important contribution, in particular since our security notion does *not* directly reduce to an encryption scheme's, and can become the basis for the development of such technologies in the future. As important is the fact that our new notion and scheme are achievable (resp., realizable) using (the practical instantiations) of basic cryptographic tools, such as a pseudorandom function generator (PRFG), a collision resistant hash function and an existentially unforgeable message authentication code (MAC) (cf. Section III).

EyeDecrypt is symmetric key-based, and the cryptographic keys needed for decryption and authentication are provisioned and directly shared between the remote content provider and the *EyeDecrypt* app running on the user's personal device (see Figure 1). In particular, the untrusted device does not have access to the keys. In the ATM scenario, for example, the keys would be shared between the bank's server, which would act as the remote content provider, and the *EyeDecrypt* app running on the user's personal device. The ATM, being untrusted, would *not* have access to the keys. Thus, the key provisioning phase is the only time at which *EyeDecrypt* requires network connectivity in order for the *EyeDecrypt* app to communicate privately with the remote content provider. For everything else, *EyeDecrypt* does not require it, making it suitable for very high-security environments where network connectivity may not be permitted. Section III-C describes the key provisioning/management aspect in more detail.

EyeDecrypt can use any type of visual data encoding (e.g., QR codes [1], Data Matrices [2], Dataglyphs [3]) for as long as this does not cause decryption failures and satisfies two basic properties (see Section III-D). In our proof of concept we opted for Dataglyphs as this particular encoding has very little structure such as no visual landmarks and no fixed block size. This gives us the flexibility of being able to change parameters of the underlying Visualizable Encryption scheme (e.g., cipher-block size) without affecting its visual encoding representation. In particular, we have developed a new dataglyph-based visual encoding scheme that can be decoded progressively, by zooming or moving the camera close to one part of the encoding, and panning to decode other parts. Due to our use of augmented reality this feels quite natural. At the same time, the security of panning becomes one of the central challenges in the design of a visualizable encryption scheme compatible with our visual encoding. In Section III-D we discuss how *EyeDecrypt* allows for the use of other visual data encodings such as QR codes.

The rest of the paper is organized as follows. In Section II we formally define our model, as well as give security definitions for *EyeDecrypt* and visualizable encryption, and specify requirements for visual encoding. In Section III we present our *EyeDecrypt* and visualizable encryption constructions, together with a proof of security, as well as key management and the dataglhyp-based visual encoding scheme. Section IV is dedicated to implementation details, including the design choices to overcome the various challenges arising from the visual encoding approach, while Section V refers to performance issues. Related work is included in Section VI. Finally, summary and conclusions are presented in Section VII.

II. Model and Definitions

In this section we present the basic model where we envision *EyeDecrypt* operating, as well as formal definitions of the different components needed for our constructions.

In its basic form, *EyeDecrypt* operates in a setting with three components, or "parties:" a user personal device U running the *EyeDecrypt* app, a server S, an a (polynomial-time) adversary Adv, controlling both the device where the information is displayed and/or entered (the "untrusted device" in Figure 1) and the shoulder-surfer(s) surrounding the user. The user device U can be any device that can capture an image, process it, and display the result to the human user. We envision the server encrypting and transmitting data to the user by visual means (e.g., rendering a visual encoding of the data on a computer screen), and the user receiving a (possibly noisy) version of that data. In turn, the user can transmit data back to the server by means of pressing buttons, active areas on a touch screen, etc., of the untrusted device. We expect the user and the server to engage in an interaction where the information transmitted at each "round" is dependent on all prior communication, as well as possibly other external factors.

In this paper we focus on *passive* adversaries who observe the visual channel, as well as other available channels of information such as the user's body language, the buttons that she presses, the areas of the touch screen that she activates, the information that is transmitted through the untrusted device, etc. (This type of adversary is also called *honest-but-curious* in the literature.) One may also consider an active adversary, which can in addition *manipulate* the communication between the server and the untrusted device, mount *man-in-the-middle* attacks, etc. (In fact, the definition of security of visualizable encryption as presented below [Definition 3] already allows for this. We leave the full treatment of active adversaries for future work.)

Data transmitted from S to U are partitioned into *frames*, and each frame is partitioned into *blocks*. The frames represent the change of the content over time, and blocks partition the frame into logical units. The choice of what constitutes a block depends on the parameters of the system. For example, a block could be a rectangular area in an image, or a group of characters in a text document.

A. Defining the EyeDecrypt activity

The security of *EyeDecrypt* is defined in a setting wherein the server can receive input from the user through the entry device or from another source such as a local hard drive or the Internet. A screen in the device is used to display information about the inputs received so far, such as outputs of a visual encoding function of the encrypted input (see below). The entry device allows the user to select values from a fixed alphabet Σ, whereas information received from other sources is viewed as arbitrary strings.

Formally, a *(stateful) EyeDecrypt scheme* is a triple of PPT (probabilistic polynomial-time) algorithms $(\mathsf{EyeDecInit}, \mathsf{EyeDecEntry}, \mathsf{EyeDecRead})$ where $\mathsf{EyeDecInit} : \mathbb{N} \to \mathcal{S} \times \mathcal{K}_{\mathsf{ED}}$ takes as input a security parameter and outputs an initial state S_0 for the *EyeDecrypt* scheme server, and a long term key for the user viewing device. Here $\mathcal{K}_{\mathsf{ED}}$ is the space of possible keys; $\mathsf{EyeDecEntry} : \mathcal{S} \times \Sigma \times \{0,1\}^* \to \mathcal{S}$ where $\mathcal{S}$ is the set of possible states of the scheme; and $\mathsf{EyeDecRead} : \mathcal{K}_{\mathsf{ED}} \times \{0,1\}^* \to \{0,1\}^*$ runs on the user device and outputs the information that is shown to the user. The expression $\mathsf{EyeDecEntry}(S, x, m)$ should be interpreted as the system receiving input x through the entry device, and receiving input m from another source.

For example, when considering a secure PIN entry application, $\Sigma = \{0, \ldots, 9\}$, and corresponding to the buttons on the keypad. In our solution for the PIN entry application, $\mathcal{S}$ will consist of the keys of the visualizable encryption scheme, and the contents displayed on the screen (more details on this in Section III-B).

We define the security of an *EyeDecrypt* scheme in terms of the information that is "leaked" to the adversary, which may vary depending on the particular real-world application that is being modeled. Specifically, the definition of security of an *EyeDecrypt* scheme is parameterized by a function $\mathsf{Leak} : \mathcal{S} \to \{0,1\}^*$ that specifies the information that is given to the adversary after each input. Looking ahead to our construction for the PIN entry case, Leak will reveal the current encrypted image displayed on the screen, as well as the number on the button that was most recently pressed by the user, but not the keys of the underlying visualizable encryption scheme. Formally, the security of an *EyeDecrypt* scheme is defined via the experiment shown in Figure 3.

$\mathsf{ExpEyeDec}(1^n, \mathsf{Adv}, \mathsf{EyeDec}, \mathsf{Leak})$:
- $S_0 \leftarrow_{\mathrm{R}} \mathsf{EyeDecInit}(1^n)$
- $((x_0, m_0), (x_1, m_1), \mathsf{st}) \leftarrow \mathsf{Adv}^{\mathsf{LeakST}}(1^n, \mathsf{Leak}(S_0))$
- where $(x_0, m_0) \neq (x_1, m_1)$
- $b \leftarrow_{\mathrm{R}} \{0,1\}$; $w = \mathsf{LeakST}(x_b, m_b)$
- $b' \leftarrow \mathsf{Adv}^{\mathsf{LeakST}}(1^n, w)$
- Output 1 if and only if $b = b'$

$\mathsf{LeakST}(x, m)$ is stateful, and works as follows:
- Initially $S \leftarrow S_0$
- Given $(x, m) \in \Sigma \times \{0,1\}^*$ do:
  - $S \leftarrow \mathsf{EyeDecEntry}(S, x, m)$
- Output $\mathsf{Leak}(S)$

Fig. 3: *EyeDecrypt security game definition*

Definition 1 *Let* $\mathsf{EyeDec} = (\mathsf{EyeDecInit}, \mathsf{EyeDecEntry}, \mathsf{EyeDecRead})$ *be an* EyeDecrypt *scheme, and* $\mathsf{Leak} : \mathcal{S} \to \{0,1\}^*$. *Then,* EyeDec *is a* Leak*-secure EyeDecrypt* scheme *if for all PPT adversaries* Adv*, and all* $n \in \mathbb{N}$*,*

$$\Pr[\mathsf{ExpEyeDec}(1^n, \mathsf{Adv}, \mathsf{EyeDec}, \mathsf{Leak}) = 1] \leq \mathsf{neg}(n).$$

Note that the algorithm EyeDecRead does not play a role in the above definition. This is because it is only used to specify the functionality of the scheme that is available to the legitimate user U (i.e., the unencrypted content from the screen). It is in fact similar to the decryption algorithm in encryption schemes.

B. Defining the building blocks

Naturally, the basic components in an *EyeDecrypt* application specify suitable ways for the information to be displayed in the rendering device and captured by the user, as well as a method for encrypting the plaintext content. Next, we elaborate on such visual encoding schemes along with some desirable properties. We then present a definition of *visualizable encryption*—a key component in our solution.

Visual encoding. Let $d_1, d_2, t_1, t_2 \in \mathbb{N}$ (see below for an explanation of these parameters), and $\mathcal{P}$ be a finite set representing possible values that can be assigned to a pixel (e.g., RGB values[2]). A *visual encoding scheme* is a pair of functions $(\mathsf{Encode}, \mathsf{Decode})$ such that $\mathsf{Encode} : (\{0,1\}^n)^{d_1 \times d_2} \to \mathcal{P}^{t_1 \times t_2}$, and $\mathsf{Decode} \equiv \mathsf{Encode}^{-1}$. $d_1 \times d_2$ is

[2]The *RGB color model* is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors.

the size of the (ciphertext) input matrix, measured in number of blocks; the size of a block is n bits. $t_1 \times t_2$ is the size (resolution) of the output (image); e.g., 640×480 pixels.

One basic but useful property of a visual encoding scheme is that it preserves the relative positioning of elements (in our case, blocks) in the source object. The following definition makes that explicit.

Definition 2 *A visual encoding scheme is said to satisfy* relative positioning *if the following conditions hold:*

1. Decode *maps* $\mathcal{P}^{\leq t_1 \times \leq t_2}$ *to* $(\{0,1\}^n)^{\leq d_1 \times \leq d_2}$*;*
2. *for all* $X \in (\{0,1\}^n)^{d_1 \times d_2}$*,* r_1 *and* r_2 *such that* $1 \leq r_1 < r_2 \leq d_1$*, and* c_1 *and* c_2 *such that* $1 \leq c_1 < c_2 \leq d_2$*, if* $Y \leftarrow \mathsf{Encode}(X)$ *and* $(r'_1, r'_2, c'_1, c'_2) = (r_1 \cdot \frac{t_1}{d_1} \ldots r_2 \cdot \frac{t_1}{d_1}, c_1 \cdot \frac{t_2}{d_2} \ldots c_2 \cdot \frac{t_2}{d_2})$*, then* $X_{r_1 \ldots r_2, c_1 \ldots c_2} \leftarrow \mathsf{Decode}(Y_{r'_1 \ldots r'_2, c'_1 \ldots c'_2})$.

Visualizable encryption. A *private-key visualizable encryption scheme* consists of a triple of PPT algorithms $\langle \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec} \rangle$, where KeyGen takes as input a security parameter $n \in \mathbb{N}$, and outputs a key; Enc takes as input a key K, a frame index f, block number i, and a plaintext m, and outputs a ciphertext; and Dec takes as input a key K, a frame index f, block number i, and a ciphertext, and outputs a plaintext.

$\mathsf{ExpVisIND\text{-}ATK}(1^n, \mathsf{Adv}, \mathsf{VisEnc})$:
$K \leftarrow_{\mathsf{R}} \mathsf{KeyGen}(1^n)$;
$(f_*, i_*, m_0, m_1, \mathsf{st}) \leftarrow \mathsf{Adv}^{\mathsf{EncATK}_K(\cdot,\cdot,\cdot), \mathsf{DecATK}_K(\cdot)}(1^n)$
$b \leftarrow_{\mathsf{R}} \{0,1\}$; $C_* \leftarrow_{\mathsf{R}} \mathsf{Enc}_K(f_*, i_*, m_b)$
$b' \leftarrow \mathsf{Adv}^{\mathsf{EncATK}_K(\cdot,\cdot,\cdot), \widehat{\mathsf{DecATK}}_K(\cdot)}(1^n)$
Let $\mathsf{view}_{\mathsf{Adv}}$ be the view of the adversary.
Output 1 if and only if $b' = b$ and $\mathsf{Check}(\mathsf{view}) = 1$.

$\mathsf{Enc}\{\mathsf{CPA}, \mathsf{CCA}\}_K(f, i, m) \stackrel{\text{def}}{=} \mathsf{Enc}_K(f, i, m)$
$\mathsf{DecCPA}_K(C) \stackrel{\text{def}}{=} \perp$
$\mathsf{DecCCA}_K(C) \stackrel{\text{def}}{=} \mathsf{Dec}_K(C)$
$\widehat{\mathsf{DecCCA}}_K(C) \stackrel{\text{def}}{=} \mathsf{Dec}_K(C)$ if $C \neq C_*$
$\perp$ if $C = C_*$
$\mathsf{Check}(\mathsf{view})$:
Let $(f_\ell, i_\ell, m_\ell)_{1 \leq \ell \leq q}$ be the queries made to EncATK. Output 1 if and only if for all ℓ, ℓ', if $f_\ell = f_{\ell'}$ and $i_\ell = i_{\ell'}$ then $m_\ell = m_{\ell'}$.

Fig. 4: *Security game definition for visualizable encryption*

Definition 3 *Let* $\mathsf{VisEnc} = \langle \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec} \rangle$*. Then,* VisEnc *is a* ATK*-secure visualizable encryption scheme, where* $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA}\}$*, if for all PPT adversaries* Adv*, all* $n \in \mathbb{N}$*,*

$$\Pr[\mathsf{ExpVisIND\text{-}ATK}(1^n, \mathsf{Adv}, \mathsf{VisEnc}) = 1] \leq \mathsf{neg}(n),$$

where $\mathsf{ExpVisIND\text{-}ATK}(\cdot,\cdot,\cdot)$ *is the experiment presented in Figure 4.*

In our proofs in Section III we require a slightly different security property from the encryption scheme, where the adversary can receive an encryption of a sequence of blocks as a challenge instead of a single block. Namely, the adversary outputs four vectors $\boldsymbol{f}_*, \boldsymbol{i}_*, \boldsymbol{m}_0, \boldsymbol{m}_1$, where $\boldsymbol{m}_0 \neq \boldsymbol{m}_1$ and receives back a vector $\boldsymbol{C}_*$ of ciphertexts of the elements of $\boldsymbol{m}_b$ with frame and block numbers at the matching positions in $\boldsymbol{f}_*$ and $\boldsymbol{i}_*$ respectively. Let us call this notions of security ATK-*secure for multiple messages*. The following claim can be shown to be true by a standard hybrid argument:

Claim 4 *Let* $\mathcal{E}$ *be an* ATK*-secure visualizable encryption scheme. Then,* $\mathcal{E}$ *is also* ATK-secure for multiple messages *with a* $\frac{1}{\nu}$ *loss in security, where* ν *is the number of messages encrypted in the challenge.*

We now provide some intuition regarding the applicability of our definition to the *EyeDecrypt* setting. Recall that a main motivation for our work is to prevent "shoulder-surfing" attacks. In such a scenario, an attacker is covertly observing the content of a (supposedly) private screen or paper document; in addition, the attacker may be able to observe the activities (gesticulation, movements, etc.) of the legitimate content's owner, and infer information. For example, by measuring how long the user spends looking at a given (encrypted) document, or the sequence of buttons that the user presses, the attacker may learn a lot about the content of the document. Our definition accounts for such a scenario similarly to the way that semantic security of encryption [4] accounts for partial knowledge of the plaintext by the adversary: by allowing the adversary in the security experiment to specify all the content but a single block in a single frame, we capture any external knowledge that the adversary may have about the plaintext.

III. CONSTRUCTIONS

A. The visualizable encryption scheme

Our main construction of a visualizable encryption scheme uses a pseudorandom function generator (PRFG), a strongly collision resistant hash function family, and an existentially unforgeable message authentication code (MAC).

Construction 1. Let F be a PRFG with key space $\mathcal{K}_{\mathsf{PRF}}$, $\mathcal{H}$ be a family of hash functions, and MAC an existentially unforgeable MAC with key space $\mathcal{K}_{\mathsf{MAC}}$. Then we construct a visualizable encryption scheme $\mathcal{E} = \langle \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec} \rangle$ as follows:

- $\mathsf{KeyGen}(1^n)$: Generate $K_{\mathsf{PRF}} \in_{\mathsf{R}} \mathcal{K}_{\mathsf{PRF}}$; $K_{\mathsf{MAC}} \in_{\mathsf{R}} \mathcal{K}_{\mathsf{MAC}}$; $H \in_{\mathsf{R}} \mathcal{H}$; and output $K = (K_{\mathsf{PRF}}, K_{\mathsf{MAC}}, H)$.
- $\mathsf{Enc}_K(f, i, M)$: Compute $C_0 \leftarrow F_{K_{\mathsf{PRF}}}(H(f, i)) \oplus M$; $\tau \leftarrow \mathsf{MAC}_{K_{\mathsf{MAC}}}(C_0)$; and output $C = (C_0, \tau, i, f)$.
- $\mathsf{Dec}_K(C)$: Interpret C as a tuple (C_0, τ, i, f), and compute $\tau' \leftarrow \mathsf{MAC}_{K_{\mathsf{MAC}}}(C_0)$. If $\tau' \neq \tau$, output $\perp$. Otherwise, compute and output $M \leftarrow C_0 \oplus F_{K_{\mathsf{PRF}}}(H(f, i))$.

Theorem 1 *The visualizable encryption scheme $\mathcal{E}$ above is CCA-secure according to Definition 3.*

Proof sketch. The proof follows by describing a sequence of hybrid arguments from the security definitions of F, $\mathcal{H}$, and MAC. We next sketch the sequence of games that gives us the proof.

- Game 0: This is the original $\mathsf{ExpVisIND\text{-}ATK}$ experiment.
- Game 1: Game 1 proceeds identically to Game 0, except that $\mathsf{Check}(\mathsf{view})$ is modified as follows. $\mathsf{Check}(\mathsf{view})'$: Proceed as in $\mathsf{Check}(\mathsf{view})$, but output 1 if and only if for all ℓ, ℓ', if $H(f_\ell, i_\ell) = H(f_{\ell'}, i_{\ell'})$ then $m_\ell = m_{\ell'}$. Game 1 and Game 0 will proceed identically, unless the adversary finds a strong collision in H.
- Game 2: Game 2 proceeds as Game 1, except that $F_{K_{\mathsf{PRF}}}$ is replaced by a random function R with the same range and domain. The fact that Game 2 and Game 1 proceed identically (except with negligible probability) follows from the pseudo-randomness of F.
- Game 3: Game 3 proceeds as Game 2, except that we further modify $\mathsf{Check}(\mathsf{view})$ to output 0 if the adversary has queried the decryption oracle on two ciphertexts $C = (f, i, C_0, \tau)$ and $C' = (f, i, C_0', \tau')$ where $(C_0, \tau) \neq (C_0', \tau')$, and both queries resulted in non-$\perp$.

This concludes the proof. ■

B. An EyeDecrypt scheme

We construct an *EyeDecrypt* scheme based on our visualizable encryption scheme $\mathcal{E}$, and the modified dataglyphs visual encoding scheme $\mathcal{V} = (\mathsf{Encode}, \mathsf{Decode})$, described in Section III-D. Our construction is parameterized by a function g that specifies how an application converts inputs to a new visual frame. Here $g(x, m, \mathsf{frame}, \pi)$ outputs a sequence of blocks $\mathsf{frame}' = (t_1, \ldots, t_n)$ that is the content of the new frame given the input from the user, an input from another source (such as a harddrive or the Internet), and the previous frame. The input π is a permutations over Σ, and its meaning will become clear in the discussion that follows the construction. To use the scheme, one only has to plug in and appropriate g into the construction below to obtain a complete scheme.

Construction 2. The generic *EyeDecrypt* scheme works as follows:

- $\mathsf{EyeDecInit}(1^n)$: Run $\mathsf{KeyGen}(1^n)$ to obtain a key K, and generate a random permutation π over Σ. Output $S_0 = (K, \pi, \perp, \perp, 0)$ and K. The two $\perp$ values in the tuple corresponds to the current cleartext and ciphertext frames, which are initially empty, and zero is the initial frame number.
- $\mathsf{EyeDecEntry}(S, x, m)$: Parse S as $(K, \pi, \mathsf{frame}, v, j)$. Generate a random permutation π' over Σ, and compute $(t_1, \ldots, t_n) \leftarrow g(\pi(x), m, \mathsf{frame}, \pi')$, set $\mathsf{frame}' = (t_1, \ldots, t_n)$, and compute $c_i \leftarrow \mathsf{Enc}_K(j, i, t_i)$ and $v' \leftarrow \mathsf{Encode}(c_1, \ldots, c_n)$. Lastly, set $S = (K, \pi', \mathsf{frame}', v', j + 1)$.
- $\mathsf{EyeDecRead}(K, v)$: Compute $(c_1, \ldots, c_n) \leftarrow \mathsf{Decode}(v)$ and $t_i \leftarrow \mathsf{Dec}_K(c_i)$ for $1 \leq i \leq n$. Output $(t_1, \ldots, t_n)$.

The intuition behind the above construction is to encrypt content as it is displayed, and to randomly permute the meaning of the possible inputs that can be received from the user input device. In the PIN entry example, we envision a touchscreen in the entry device where the nine digits are randomly re-ordered each time the user enters a PIN digit. Alternatively, the device may have a keypad with unlabeled buttons, and a random mapping of buttons to digits will be displayed to the user in encrypted form. Before proceeding with a formal description of the PIN entry solution, we prove the security of the above construction.

Theorem 2 *Let* $\mathsf{Leak}(S) = v$*. Then, the* EyeDecrypt *scheme above is* Leak*-secure according to Definition* 1 *if* $\mathcal{E}$ *is a* CPA*-secure visualizable encryption scheme.*

Proof sketch. We prove the theorem by reducing the security of *EyeDecrypt* to the security of $\mathcal{E}$. Let Adv be an adversary that breaks Leak-security of *EyeDecrypt*. Then, we construct Adv' that breaks the CPA-security for multiple messages of $\mathcal{E}$. Then, by Claim 4, we obtain the security of *EyeDecrypt*.

Our adversary Adv' works as follows. Initially, Adv' simulates $\mathsf{EyeDecInit}(1^n)$ except that no encryption key is generated. Adv' then simulates Adv. To answer a query (x, m) to the LeakST oracle, Adv' works as follows. Adv' computes $(t_1, \ldots, t_n) \leftarrow g(\pi(x), m, \mathsf{frame}, \pi')$, and obtains $c_i = \mathsf{Enc}_K(j, i, t_i)$ for $1 \leq i \leq n$ by querying its EncCPA oracle. All other steps are identical to EyeDecEntry. Adv' then computes and returns $v' \leftarrow \mathsf{Encode}(c_1, \ldots, c_n)$ to Adv.

When Adv submits the challenge tuple $(x_0, m_0), (x_1, m_1)$, Adv' computes $\mathsf{frame}_b \leftarrow g(\pi(x_b), m_b, \mathsf{frame}, \pi')$ for $b \in \{0, 1\}$. If $\mathsf{frame}_0 = \mathsf{frame}_1$, then Adv' gives up, and outputs a random bit. Otherwise, Adv' submits $(\mathsf{frame}_0, \mathsf{frame}_1)$ as its challenge in the ExpVisIND-CPA experiment. Given a vector of ciphertexts $(c_1^*, \ldots, c_n^*)$, Adv' constructs the challenge ciphertext as above, and returns the encoded version to Adv. The simulation is concluded naturally.

Given the above construction, Adv' simulates Adv perfectly in the ExpEyeDec experiment, except when $\mathsf{frame}_0 = \mathsf{frame}_1$. However, in this case, Adv obtains no information about b in the challenge. Therefore, Adv' wins with the same advantage as Adv. ■

Instantiating *EyeDecrypt* for secure PIN entry. We are now ready to provide a complete solution to the PIN entry application. Given Construction 2, the missing piece is the function g that defines the functionality of the application. The exact nature of g will depend on the content being protected by the PIN. However, any PIN protected application must allocate some of the output blocks of g to display a permuted numeric keypad. Suppose that the user input device is a (fixed) numeric keypad, and suppose that blocks $t_1, \ldots, t_{n'}$ in the plaintext visual frame are allocated to the permuted keypad. Let $P(\mathsf{pin}, \mathsf{data})$ be a program that, given the PIN and additional data, generates blocks $t_{n'+1}, \ldots, t_n$. Then, $g(\mathsf{pin}, \mathsf{data}, \mathsf{frame}, \pi)$ computes $(t_{n'+1}, \ldots, t_n) \leftarrow P(\mathsf{pin}, \mathsf{data})$, and computes blocks $t_1, \ldots, t_{n'}$ by generating an image that shows the digit d written on the physical button that has the digit label $\pi^{-1}(d)$.

C. Key establishment

As mentioned in Section I and made evident by the definitions and constructions above, *EyeDecrypt* is symmetric key-based. For completeness we briefly sketch here how the personal device running the *EyeDecrypt* app and the content generating server are able to share a cryptographic key in a secure manner. The personal device is provisioned with a master key using standard methods (e.g., by performing a key exchange with the content provider on a connection that is authenticated with the user's credentials). Then, whenever a new content is being viewed by the user, it will first contain a visually (e.g., QR-, Dataglyph-) encoded nonce; by applying a key-derivation function (KDF) to the per-document nonce the viewer and the content provider are able to derive the same key from the shared master key.[3]

D. A dataglyphs-based visual encoding scheme

In this section we present the basis of our visual encoding scheme. Further details are provided in the next section.

Nowadays many visual encoding solutions exist. QR Codes, Data Matrixes and Microsoft High Capacity Color Barcodes (HCCB) [5] are just a few examples.

[3]Recall that the focus of this paper is on passive adversarial attacks. Naturally, in the case of active attacks, additional precautions need to be taken (e.g., to guarantee freshness, avoid replay attacks, etc.).

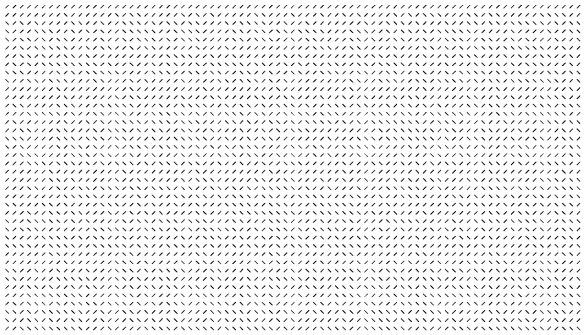


Fig. 5: *Dataglyphs encoding*

EyeDecrypt is not bound to one specific visual encoding algorithm. It does require, however, that two properties be satisfied.

1. Locality: cropped visual encoding correctly decodes to a sub-matrix of input.

2. Relative positioning: adjacent input sub-matrixes are adjacent in the output of the visual encoding (cf. Definition 2).

Locality is important as it guarantees that the decryption operation is not impaired by some limitation of the visual encoding algorithm. For example, let us consider one QR code encoding two blocks of ciphertext. If the phone camera is positioned so as to capture just a part of the QR code, decoding will fail. It will fail even though the part of QR code captured by the camera contained one full block of ciphertext which could have been decrypted without any problem. QR codes clearly do not satisfy the first property when multiple blocks of ciphertexts are encoded into a single QR code. This means that in order to use QR codes with *EyeDecrypt*, each QR code must encode one single block of ciphertext and nothing more.

The second property ensures that the visual encoding algorithm preserves the spatial ordering of the blocks of ciphertext (which corresponds to the spatial ordering of the plaintext). This is needed for the panning and zooming functionality of our augmented reality scheme: as the user pans, the information that we can display scrolls appropriately, and as we zoom out, the additional blocks that we can decrypt visually extend the existing decrypted blocks.

Dataglyphs [3] (see Figure 5) are a well known visual encoding algorithm that satisfies the two requirements above. In particular, each dataglyph represents one to a few bits of information and it can be read independently from the other dataglyphs since, unlike QR codes, there is no notion of a block. Furthermore, dataglyphs do not require any visual landmark, which saves space for the actual content. We decided to use dataglyphs as the extra flexibility they provide allowed us to modify some parameters of the underlying Visualizable Encryption scheme (e.g., cipher-block size) without affecting its visual encoding representation. This made experimentation with different visualizable encryption parameters much easier. In Section III-B we referred to the dataglyph visual encoding scheme as $\mathcal{V} = (\mathsf{Encode}, \mathsf{Decode})$.

As mentioned earlier, in our current construction ciphertext is visually encoded using dataglyphs. In particular, each dataglyph can have an inclination equal to one of two possible angles that is, -45° and $+45^\circ$. An angle of -45° corresponds to the bit 1 and an angle of $+45^\circ$ corresponds to the bit 0. Although these are the only two values used, because of noise introduced by the phone camera, different angle values may be detected. Dataglyphs with values different from the above, however, either get discarded or get mapped to one of the two correct values. Using two angle values very distant from each other such as the ones we picked, makes the system more resilient to noise.

IV. Implementation Details

We envision *EyeDecrypt* to be a natural fit for augmented-reality devices such as Google Glass [6], where the user will be able to just look at encrypted content and get the decrypted version displayed directly on the screen of the glasses. However, given that such devices are not yet widely available, we implemented *EyeDecrypt*

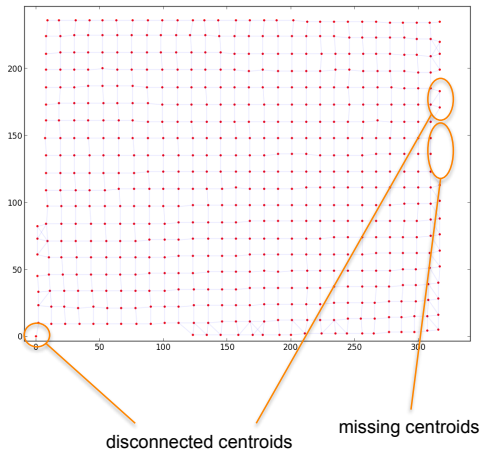
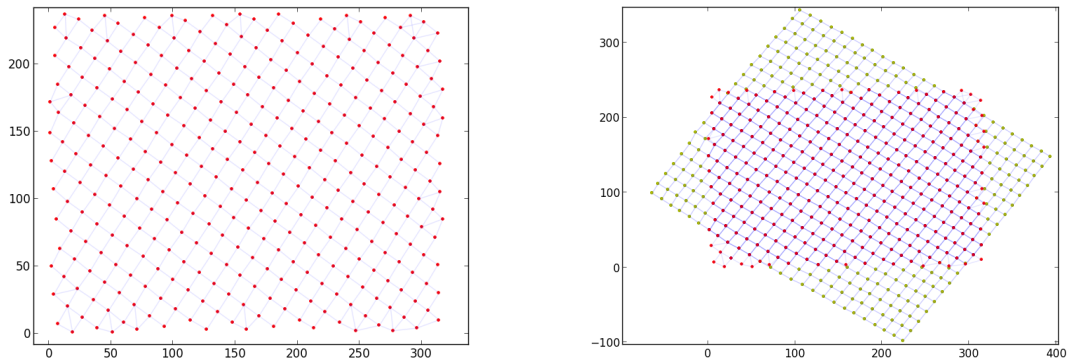


Fig. 6: *Noise and artifacts in the decoding process*



(a) Rotated centroids

(b) Rebuilt graph

Fig. 7: *Rebuilding a graph from noisy centroids with unknown rotation*

as an Android app running on a Samsung Galaxy S4 cellphone with 2 GB of RAM, a Quad-core 1.9 GHz Krait 300 processor and running Android version 4.2.2 (Jelly Bean). Furthermore, the phone was equipped with an 13 MP rear-facing camera. We used OpenCV version 2.4.5 [7] as the Android computer-vision library in order to manipulate and process images from the phone camera.

In order to provide an acceptable user experience, the *EyeDecrypt* app does not expect users to take a snapshot of the encrypted content but, rather, it shows a "live" camera view. The app processes "live" camera frames on the fly and shows the decrypted content overlaid on the camera view itself. The only action users need to perform is to position the phone camera in front of the encrypted document they wish to decrypt and move the camera around to decrypt different parts of the document.

Note that visualizable encryption is not technically needed if the amount of content that we need to encrypt is very small, for example, if it fits within a single QR code. Visualizable encryption is meant to be useful regardless of the size of the plaintext/ciphertext. Thus *EyeDecrypt* does not require ad-hoc handling of special cases, and works the same for small amounts of ciphertext as well as very large amounts of ciphertext.

In the current implementation we encrypt the plaintext instantiating the PRF in the visualizable encryption scheme of Section III-A with AES-128, and visually-encode it using dataglyphs. Currently, no MAC is implemented and therefore our implementation is only CPA secure.

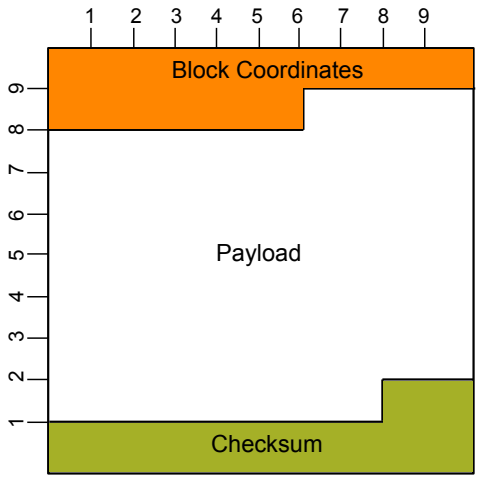


Fig. 8: *Ciphertext block structure*

Figure 5 shows an example of an encrypted document used in our prototype. As we can see, unlike QR codes, HCCB and other visual encoding techniques, there are no visual landmarks present in the visual encoding. Still, we make no assumption on the angle at which the user holds the phone; that is, decryption should not fail if the user holds the phone rotated to a certain angle. The only assumption we make is that the user holds the phone so as to include at least one full block of ciphertext in the camera view.

We now list the steps that *EyeDecrypt* goes through in order to decode a visually encoded image of ciphertext.

Image format conversion. In Android there are two default image formats that need to be supported by all phones that is, NV21 and YV12[4]. When the EyeDecrypt app gets a camera frame, it converts its pixels values represented in one of the formats above into an (R, G, B) representation so to better manipulate the resulting image.

Removing Moiré patterns. As mentioned in the introduction, *EyeDecrypt* is meant to work on a traditional medium like paper but also on an electronic medium like a computer screen. In this second scenario, additional noise is introduced to the image captured by the phone camera. In particular, Moiré patterns [8] are well-known artifacts present in digital images. In order to reduce Moiré patterns when reading a visual encoding from a screen, we apply a series of low-pass filters and high-pass filters to filter out such patterns as much as possible while, at the same time, trying to enhance the dataglyphs. In our implementation we use OpenCV and in particular, we use a Gaussian Blur as low-pass filter and a Laplacian as high-pass filter.

Edge detection. The image is then converted to gray scale in order to perform edge detection on the dataglyphs. In particular, we use the Scharr transform to perform edge detection.

Contours. We further process the output of the edge detection algorithm to compute the contour of each dataglyph. To achieve this, standard computer vision techniques can be applied using the OpenCV library. For each contour, we calculate its centroid coordinates and angle.

Building the graph. By leveraging the centroids coordinates of dataglyphs, we build a graph having the centroids as vertices. In particular, we perform the Delaunay Triangulation on the set of centroids and remove the extra edges between centroids so that each centroid is connected only to its four closest neighbors. This graph representation is useful to overcome various problems due to noise and other factors that we describe below.

Removing noise. Camera lens deformation, non-uniform light conditions, variable distance from content and camera resolution all lead to the creation of noise and artifacts in the detection of the contours that is, in the detection of the centroids. Such artifacts are usually located at the edges of the camera field of view which translates to

[4]This format is guaranteed to be supported starting with API level 12.

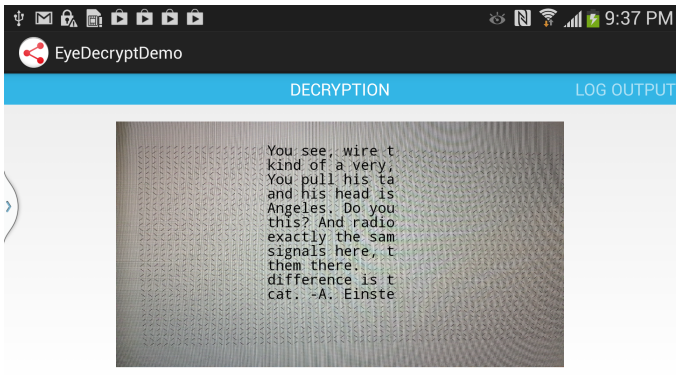


Fig. 9: *Decoding and decryption of 1,834 bits of ciphertext*

disconnected or missing centroids at the edges of the graph. Figure 6 shows this problem. We can see the presence of partially disconnected centroids as well as several centroids missing from the graph.

The way users hold their phone represents another significant source of noise. In particular, given that the visual encoding we use does not have any landmark to help with alignment, if the phone is rotated by a significant amount, it may be very hard to tell left from right and top from bottom of the visually encoded content captured by the camera. Figure 7a shows this problem. As we can see, by just looking at the centroids of the contours we cannot tell the correct alignment of the ciphertext.

In order to solve all these issues, we apply various graph-theory algorithms that allow us to remove all the artifacts due to noise and reconstruct the graph with the correct rotation modulo 90°. Figure 7b shows the graph reconstructed from the centroids shown in Figure 7a. We can see that we were able to remove artifacts and infer the camera rotation modulo 90° which is an essential step to correctly decode the content.

Decoding. The corrected graph from the previous step is next converted to a binary matrix by converting the angle information of each centroid into ones and zeros.

In the current implementation, one block of ciphertext has dimensions 10×10 bits. The first 16 bits of the block represent the coordinates i and j of that block in the frame, while the last 12 bits of the block represent a checksum. This checksum is the truncated output of AES-128 applied to the coordinates i, j of the block. We know we have found a valid block of ciphertext when computing AES-128 over the first 16 bits of a block, we get the same checksum found in the last 12 bits of that block. If the checksum fails, we move one column to the right in the matrix and perform the same check on the new block until we have tested the whole matrix. If a valid block is found, each block of the matrix is decrypted. Finally, the decrypted content from all the decrypted blocks is displayed as an overlay on the phone camera view; see Figure 9. If no valid blocks are found, we output an error message to the user indicating a decryption failure.

V. Performance Evaluation

As mentioned in the previous section, each block of ciphertext has a dimension of 10×10 bits. Figure 8 shows the structure of the block. As we can see, the first 16 bits are used to encode the coordinates i, j corresponding to the position of the block in the document, while the last 12 bits are used for the block checksum. This leaves 72 bits of encrypted payload per block. Given that the visualizable encryption scheme uses a one-time pad (see Section III-A), we can encrypt 72 bits of data in each block. In the case of text, this means that we can encrypt nine characters per block of ciphertext.

In general, users will hold their device so that multiple blocks will be decoded at once, that is, a multiple of nine characters will be displayed at once to the user. Increasing the ciphertext block size would reduce the overhead due to block coordinates and checksum. A larger block size, however, would also mean that users have to hold their devices at a larger distance from the encoded image in order to fit at least one block in the camera field of view. The larger distance would add additional noise, possibly leading to a higher probability of decryption failure.

Figure 9 shows the correct decoding and decryption of 1,834 bits of ciphertext displayed on a computer screen. As we can see, the decoding was successful despite the presence of Moiré patterns. In such case, the decoding took

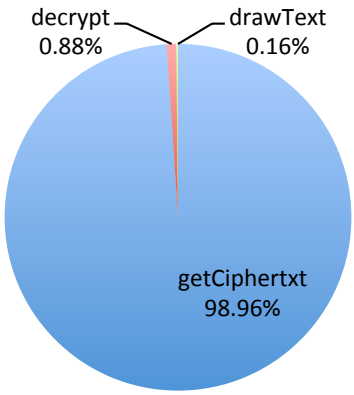


Fig. 10: *Contributions to the average processing time over 50 experiments for decoding and decrypting a numerical keypad*

an average time of 1.5 seconds. This time, however, largely depends on the number of cipher-blocks in the frame. In the case of a numerical keypad for example, the visual encoding includes 12 cipher-blocks and we are able to process the whole encoded keypad at a rate of one frame per second or higher.

Figure 10 shows the contribution of the main components of *EyeDecrypt* to the total processing time required for decoding and decrypting a keypad displayed on a computer monitor. In particular, we break the total computational time in three main components that is, `getCiphertxt`, `decrypt` and `drawText`. The `getCiphertxt` component is the time required to decode the image and build a matrix of bits representing the corresponding ciphertext, the `decrypt` component is the time required for the actual decryption of the matrix of ciphertext and the `drawText` component is the time required to display the plaintext in the phone camera view. As we can see, most of the time is spent for decoding the image and build the matrix of ciphertext while the actual decryption of the ciphertext and drawing operations take a negligible amount of time. In particular, while decoding and building the matrix of ciphertext took on the order of 1 second, actual decryption took a time on the order of a few hundred milliseconds while drawing the plaintext over the camera view took only a few milliseconds.

The reason why `getCiphertxt` takes such a long time is shown in Figure 11. In particular, of all the steps involved in the decryption process (see Section IV), "Building the graph" takes most of the time followed by the combination of "Removing noise" plus "Decoding". During these three steps we perform the Delaunay triangulation on the set of centroids and then further process it to remove noise and extract its graph representation. From such graph we then build the binary matrix of ciphertext. In Figure 11 we see that running the Delaunay triangulation takes 55% of the time while processing its output to remove noise, build the graph and construct the matrix takes another 23%. The remaining amount of time of `getCiphertxt` accounts for all other steps as described in Section IV. In particular, converting the image to its RGB representation takes only 4% of the time while edge detection of dataglyphs and finding contours takes 3% of time each. Removing Moiré noise takes 6% of the time as does processing the contours of the dataglyphs.

The visual encoding based on dataglyphs could be easily enhanced to increase the amount of information it conveys. As we mentioned in Section III-D, we currently use only two angle values to encode ones and zeros—namely, -45° for bit 1 and $+45^\circ$ for bit 0.

Naturally, the problem in having only two possible values is that less information can be conveyed in the dataglyph encoding. In particular, by using 0°, $+45^\circ$, -45° and 90°, each dataglyph could encode two bits of information. This, however, would make dataglyphs less resilient to noise. Other ways to enhance dataglyphs would be by using different colors and sizes so that for each dataglyph we can now specify angle, color and size. If we assume four possible angles, four independent colors and two different sizes, one dataglyph could now convey 5 bits of information. Enhancing the visual encoding so to convey more information as described above is left for future work.

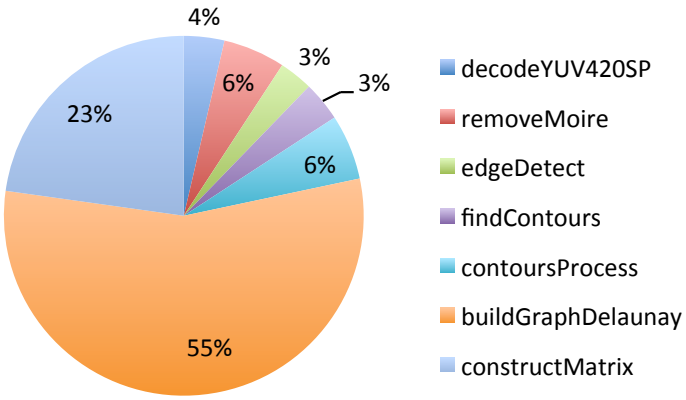


Fig. 11: *Percentage of time required by each step of the algorithm used to decode a visually encoded image of ciphertext (getCiphertxt)*

VI. Related work

Human-computer interactions almost universally involve human vision, and so *EyeDecrypt* or any other HCI technology is subject to the security limitations of vision. In particular, vision is susceptible to interception, by means as simple as shoulder surfing or as sophisticated as capturing the reflection of images from the surface of the eye [9], [10], [11]. *EyeDecrypt* protects against many interception attacks by encrypting the visual channel between the encoding and a personal device. The visual channel from the personal device to the eye remains unprotected by *EyeDecrypt* itself; *EyeDecrypt* is intended for use only when the personal device remains inaccessible to adversaries. When eye reflections are a concern but it is still desirable to use the visual channel, we know no protection short of an enclosed display. *EyeDecrypt* is compatible with such a display.

On the other hand, vision has some inherent security advantages. Humans generally know with certainty what physical object they are looking at (as opposed to what device has sent them a wireless transmission), and vision is resistant to active tampering. For example, there are a few techniques to overload camera sensors but a user can detect this by comparison with their own vision. Consequently, visual encodings are widely use in security for key establishment, wireless device pairing, and trusted display establishment [12], [13], [14], [15]; *EyeDecrypt* can be used for these purposes as well.

Computer vision researchers have been studying visual encodings for decades, seeking to increase their capacity and their robustness against lens distortion, image compression, non-uniform lighting, skew, motion blur, shake, low camera resolution, poor focus, etc. [16], [17], [18]. Techniques for zooming into visual encodings include recursive grids of barcodes [19] and nested barcodes [20]. Fourier tags [21] handle the opposite case of zooming out: at close distance, they can be completely decoded, but as the camera zooms out, fewer bits can be decoded; low-order bits are lost before high-order bits.

Encrypted content is often used in single barcodes (e.g., [22]) but less often in multiple barcodes. Fang and Chang describe a method for encoding a large encrypted message in multiple barcodes [15]. All barcodes must be decoded and decrypted to view the message, and the barcodes must be presented in a known order, unlike our blocks which can be viewed independently. They are concerned with *rearrangement attacks* in which the adversary is able to rearrange the order of the barcodes so that the message cannot be decrypted. Their solution is to use a visual cue (numbers in sequence) over which each barcode is interleaved. The user can manually verify that the barcodes are in the correct order, while the device handles the decoding and decryption of the actual data. In our solution, visual clues are not necessary, as the frame and block numbers of adjacent regions are directly encoded and can be read and compared automatically by the device.

Many defenses against shoulder surfing during PIN entry have been proposed; we discuss a representative sampling here. We emphasize that unlike *EyeDecrypt*, these systems do not encrypt the device display, so they are not appropriate for *displaying* private information, only entering it.

EyePassword [23] and Cued Gaze-Points [24] are two systems that use gaze-based input for password/PIN entry.

These systems require the public device (e.g., an ATM) to have a camera and they work by computing the point on the screen that the user is looking at. In the Cued Gaze-Points system PIN entry works by the user selecting gaze-points on a sequence of graphic images. In EyePassword, the user gazes at a standard onscreen keyboard. The key assumption in both cases is that the adversary does not see the input at all (the adversary does not have a view of the user's eyes). In contrast, *EyeDecrypt* assumes that the adversary can see the input but only in obfuscated form (randomized and encrypted).

Roth *et al.* [25] require users to enter PINs via *cognitive trapdoor games*, e.g., a sequence of puzzles that are easy to solve (by an unaided human) with knowledge of the PIN, but hard to solve without it. Their scheme emphasizes useability and is intended only to defend against attackers that are unaided humans (for example, with human short-term memories). Unlike gaze-entry systems, and similar to *EyeDecrypt*, it can work without modifying ATM hardware.

In the ColorPIN system [26], a user's PIN is a sequence of colored digits, and the ATM displays a ten-digit keypad where each digit appears with three colored letters. For example, the digit "1" could appear above a black "Q," a red "B," and a white "R," and the user would enter "black 1" by hitting "Q." Each letter appears with multiple digits, so that a sequence of letters is associated with multiple sequences of digits. A shoulder-surfing observer thus gets partial information about the PIN (e.g., for a four-digit pin the observer knows that it is one of $3 \times 3 \times 3 = 81$ possibilities). *EyeDecrypt*'s visual encryption protects against this sort of leakage.

MobilePIN [27], like *EyeDecrypt*, uses a trusted personal device to aid PIN entry. In MobilePIN, the ATM displays its wireless address and authentication token onscreen as a QR code, and the user reads the code using the camera of the personal device. The personal device can then establish a secure wireless connection between with the ATM (secure pairing using the visual channel). The user enters her PIN on the trusted personal device, which transmits it to the ATM over the wireless channel. MobilePIN therefore has similar assumptions as *EyeDecrypt*, except in addition it assumes that the ATM is equipped with a radio.

Most other shoulder-surfing-resistant PIN entry methods involve changing the authentication process, for example by using graphical passwords or security tokens, or by requiring network connectivity or device pairing.

Finally, *EyeDecrypt* has the additional ability to ensure that only legitimate users can view the information that is openly displayed, and in that sense bears some similarity to broadcast encryption ([28] and numerous follow-ups), with closely related applications such as pay-TV. A fundamental difference in *EyeDecrypt* is the public-view nature of the rendering device.

VII. Summary and Conclusions

In this paper we introduced *EyeDecrypt*, a novel technology aimed at protecting content displayed to the user as well as interactions the user has with the system (e.g., by typing). In particular, we do not trust the device the user interacts with as it may have been compromised (e.g., root-kits, key-loggers) nor do we trust the user's environment. ("shoulder surfing"). Although we presented a visual encoding scheme based on Dataglyphs, *EyeDecrypt* can easily support any other visual encoding. It protects electronic content (e.g., displayed on a computer screen or a smartphone display) as well as non-electronic content (e.g., passports, medical records). *EyeDecrypt* does not require network connectivity to operate except for the initial key exchange with the content provider.

We envision *EyeDecrypt* being applied to very different scenarios. For example, *EyeDecrypt* can protect interactions users have with their laptops or smartphones (i.e., running the *EyeDecrypt* app on Google Glass); *EyeDecrypt* can protect users when buying in stores using mobile payment platforms (e.g., Square) where they have to input their PIN on a store device. Similarly, *EyeDecrypt* could be used to encrypt personalized ads in public settings (e.g., showing personalized ads in a movie theater) without violating users privacy. If the human eye can see it, *EyeDecrypt* can protect it.

References

[1] ISO, "Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification," International Organization for Standardization, Geneva, Switzerland, ISO 18004:2006, 2006.

[2] ——, "Information technology – Automatic identification and data capture techniques – Data Matrix bar code symbology specification," International Organization for Standardization, Geneva, Switzerland, ISO 16022:2006, 2006.

[3] R. F. Tow, "Methods and means for embedding machine readable digital data in halftone images," May 24, 1994, US Patent 5,315,098.

[4] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022000084900709

[5] D. Parikh and G. Jancke, "Localization and segmentation of a 2D high capacity color barcode," in *IEEE Workshop on Applications of Computer Vision*. IEEE, 2008, pp. 1–6.

[6] Google, "Google Glass." [Online]. Available: http://www.google.com/glass

[7] itseez, "Open Source Computer Vision (OpenCV) Library." [Online]. Available: http://opencv.org

[8] Wikipedia, "Moiré pattern," 2013. [Online]. Available: http://en.wikipedia.org/wiki/Moir%C3%A9_pattern

[9] K. Nishino and S. K. Nayar, "Corneal imaging system: Environment from eyes," *International Journal of Computer Vision*, vol. 70, no. 1, pp. 23–40, 2006.

[10] M. Backes, T. Chen, M. Drmuth, H. P. A. Lensch, and M. Welk, "Tempest in a teapot: Compromising reflections revisited," in *IEEE Symposium on Security and Privacy*, 2009, pp. 315–327.

[11] M. Backes, M. Drmuth, and D. Unruh, "Compromising reflections-or-how to read LCD monitors around the corner," in *IEEE Symposium on Security and Privacy*, 2008, pp. 158–169.

[12] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *IEEE Symposium on Security and Privacy*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 110–124.

[13] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Secure device pairing based on a visual channel," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 306–313.

[14] G. Starnberger, L. Froihofer, and K. Goeschka, "QR-TAN: secure mobile transaction authentication," in *International Conference on Availability, Reliability and Security, 2009. ARES '09*, 2009, pp. 578–583.

[15] C. Fang and E.-C. Chang, "Securing interactive sessions using mobile device through visual channel and visual inspection," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 69–78. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920272

[16] J. Liang, D. Doermann, and H. Li, "Camera-based analysis of text and documents: a survey," *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 7, no. 2-3, pp. 84–104, Jul. 2005. [Online]. Available: http://link.springer.com/article/10.1007/s10032-004-0138-z

[17] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007.

[18] S. D. Perli, N. Ahmed, and D. Katabi, "PixNet: interference-free wireless links using LCD-camera pairs," in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '10. New York, NY, USA: ACM, 2010, pp. 137–148. [Online]. Available: http://doi.acm.org/10.1145/1859995.1860012

[19] D. Reilly, H. Chen, and G. Smolyn, "Toward fluid, mobile and ubiquitous interaction with paper using recursive 2D barcodes," in *3rd International Workshop on Pervasive Mobile Interaction Devices (PERMID 2007)*, May 2007.

[20] K. Tateno, I. Kitahara, and Y. Ohta, "A nested marker for augmented reality," in *IEEE Virtual Reality Conference, 2007. VR '07*, 2007, pp. 259–262.

[21] J. Sattar, E. Bourque, P. Giguere, and G. Dudek, "Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction," in *Fourth Canadian Conference on Computer and Robot Vision, 2007. CRV '07*, 2007, pp. 165–174.

[22] D. Conde-Lagoa, E. Costa-Montenegro, F. Gonzalez-Castao, and F. Gil-Castieira, "Secure eTickets based on QR-Codes with user-encrypted content," in *2010 Digest of Technical Papers International Conference on Consumer Electronics (ICCE)*, 2010, pp. 257–258.

[23] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd, "Reducing shoulder-surfing by using gaze-based password entry," in *Proceedings of the 3rd Symposium on Usable Privacy and Security*, ser. SOUPS '07. New York, NY, USA: ACM, 2007, pp. 13–19. [Online]. Available: http://doi.acm.org/10.1145/1280680.1280683

[24] A. Forget, S. Chiasson, and R. 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*. New York, NY, USA: ACM, 2010, pp. 1107–1110. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753491

[25] V. Roth, K. Richter, and R. Freidinger, "A PIN-entry method resilient against shoulder surfing," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2004, pp. 236–245. [Online]. Available: http://doi.acm.org/10.1145/1030083.1030116

[26] A. De Luca, K. Hertzschuch, and H. Hussmann, "ColorPIN: Securing PIN entry through indirect input," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2010, pp. 1103–1106. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753490

[27] A. De Luca, B. Frauendienst, S. Boring, and H. Hussmann, "My phone is my keypad: Privacy-enhanced PIN-entry on public terminals," in *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group*. New York, NY, USA: ACM, 2009, pp. 401–404. [Online]. Available: http://doi.acm.org/10.1145/1738826.1738909

[28] A. Fiat and M. Naor, "Broadcast encryption," in *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '93. London, UK, UK: Springer-Verlag, 1994, pp. 480–491. [Online]. Available: http://dl.acm.org/citation.cfm?id=646758.705697