EyeDecrypt — Private Interactions in Plain Sight

Andrea Forte Juan Garay Trevor Jim Yevgeniy Vahlis
AT&T Labs
Security Research Center
{forte, juan.a.garay, trevor, evahlis}@att.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 public 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), without revealing the details of the interaction to others who may be watching.

The user views 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 decrypted data is displayed as an image overlay on the personal device—a form of augmented reality. The user's inputs are protected through randomization.

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

In this paper we introduce *EyeDecrypt*, a technology that enables only selected users to (intelligibly) view information displayed in public on electronic displays or printed surfaces. In addition, users are able to (privately) interact with the system and provide input through a publicly-viewable touch screen or keyboard.

In *EyeDecrypt*, the data is viewed on a closely-held personal device, like a pair of smart glasses with a camera and heads-up display, or a smartphone. Just as in human vision, *EyeDecrypt* employs fluid panning and zooming to take in a large "scene." The decrypted data is displayed as an image overlay on the camera view of the personal device—a form of augmented reality. The image overlay can tell the user how to interact privately with the system, for example by labeling the physical keys on the ATM keypad in a random order. Usability is retained since the user sees herself pressing the labeled keys through the augmented camera view.

The purpose of *EyeDecrypt* is of course to protect against *shoulder surfing*, in which an "adversary" (say, a next-seat neighbor) is able to glean private information from the user's display and inputs. As an illustrative example, shoulder surfing attacks directed at automated teller machine fraud are increasingly common and sophisticated. In these attacks, the adversary modifies a standard ATM with a camouflaged card reader and camera. When Alice inserts her card into the ATM, the card reader records its data, and the camera records Alice entering her PIN. The information can be stored for later pickup or transmitted wirelessly; eventually, the adversary uses the data to duplicate Alice's card, and uses the duplicate card and Alice's PIN to withdraw funds.

EyeDecrypt can prevent PIN compromise, as follows. The bank and Alice's device share a secret key that is used to ("visually") encrypt a series of frames displayed on the ATM; these frames reveal nothing to the adversary's camera. Alice, however, can read the ATM display by pointing her personal device at

the display. Her device, which is configured with the shared secret key, decrypts the frame and overlays Alice's camera image with the decrypted image on Alice's display. For example, Alice's display could show the ATM keypad with its keys overlaid with the digits 0–9 in a randomized order. Alice can then enter her PIN on the ATM keypad without revealing it to the adversary[1], even though the adversary's camera captures the physical keys she enters on the ATM keypad. The adversary has captured Alice's card data but not her PIN, and hence cannot withdraw her funds.

Notice that Alice's device only needs to share a key with the bank, and *not* the ATM. This is important because there are many thousands of ATMs, and they are publicly accessible and hence vulnerable. The shared key can be established when Alice obtains her card at the bank, and cards can tell the ATM whether *EyeDecrypt* should be used so that customers both with and without smart glasses can use the same ATM (opt-in, incremental deployment). *EyeDecrypt* can work on existing ATMs without physical modification due to our use of augmented reality.

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.

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. Additionally, our notion of security must 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. Formally defining the security of this new applications is an important contribution, in particular since our security notion does not directly reduce to an encryption scheme's, for example, and can become the basis for the development of such technologies in the future.

Visual data encodings commonly available on smart devices, such as QR codes [9], either have insufficient capacity or are otherwise unsuitable for visualizable encryption needs. Therefore, we have developed a new 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.

PIN entry on an untrusted device is a demanding application that uses all of *EyeDecrypt*'s components. We can think of many other less-demanding applications of *EyeDecrypt*, some of which can dispense with one or more components. For example, when using a laptop on a plane, we might assume that the laptop is highly trusted and has an encrypted wireless channel to the smart glasses; in this case we may not need the laptop display at all, but *EyeDecrypt*'s augmented reality and randomized input entry would still be useful for entering passwords and other sensitive information via the laptop keyboard. Or, we may want to use *EyeDecrypt*'s encryption and encoding with non-interactive paper documents. Some other applications might include a physician reviewing a patient's medical record on a public display in a medical office, or a hierarchical organization where different levels of information confidentiality are required for the same document[2].

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 a dataglhyp-based [22] 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. Finally, related work is included in Section VI.

[1]In fact, the keypad layout is re-randomized after each digit entry.

[2]In this more advanced application, *EyeDecrypt* would support different levels of visualization of the same object.

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 construction.

In its basic form, *EyeDecrypt* operates in a setting with three parties: a user viewing device U, server S, and adversary Adv. The user device can be any device that can capture an image, process it, and display the result to the human user. We envision the server 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, voice commands, active areas on a touch screen, or another input 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, etc. One may also consider an active adversary, which can in addition manipulate the communication between the server and the user. This could occur, for example, if the user is interacting with a terminal infected by malware that is displaying information that is transmitted by a remote trusted server. (In fact, the definition of security of visualizable encryption as presented already allows for this.)

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 entry device or from another source such as a local hard drive or the Internet. A screen is then 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 1.

$\mathsf{ExpEyeDec}(1^n, \mathsf{Adv}, \mathsf{EyeDec}, \mathsf{Leak})$:

$S_0 \leftarrow_{\mathrm{R}} \mathsf{EyeDecInit}(1^n)$

$((x_0, m_0), (x_1, m_1), \mathsf{st}) \leftarrow \mathsf{Adv}^{\mathsf{LeakST}}(1^n, \mathsf{Leak}(S_0))$

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. 1: *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[3]). 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 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* $X_{r_1 \ldots r_2, c_1 \ldots c_2} \leftarrow \mathsf{Decode}(Y_{r'_1 \ldots r'_2, c'_1 \ldots c'_2})$.

[3]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.

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.

$\mathsf{ExpVisIND\text{-}ATK}(1^n, \mathsf{Adv}, \mathsf{VisEnc})$:
$K \leftarrow_{\mathrm{R}} \mathsf{KeyGen}(1^n)$;
$(f_*, i_*, m_0, m_1, \mathsf{st}) \leftarrow \mathsf{Adv}^{\mathsf{EncATK}_K(\cdot,\cdot,\cdot), \mathsf{DecATK}_K(\cdot)}(1^n)$
$b \leftarrow_{\mathrm{R}} \{0,1\}$; $C_* \leftarrow_{\mathrm{R}} \mathsf{Enc}_K(f_*, i_*, m_b)$
$b' \leftarrow \mathsf{Adv}^{\mathsf{EncATK}_K(\cdot,\cdot,\cdot), \widehat{\mathsf{DecATK}}_K(\cdot)}(1^n)$
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. 2: *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 2.*

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 [6] 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 $K_{\mathsf{PRF}} \in_{\mathrm{R}} \mathcal{K}_{\mathsf{PRF}}$; $K_{\mathsf{MAC}} \in_{\mathrm{R}} \mathcal{K}_{\mathsf{MAC}}$; $H \in_{\mathrm{R}} \mathcal{H}$; 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-C. 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 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. 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.

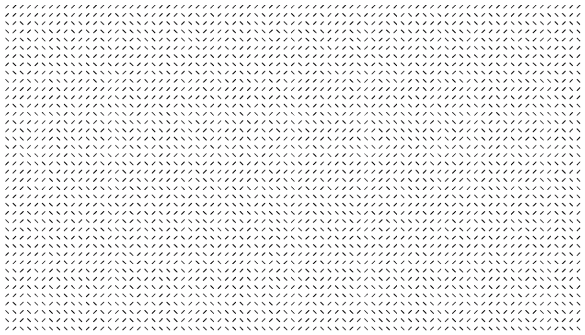


Fig. 3: *Dataglyphs encoding*

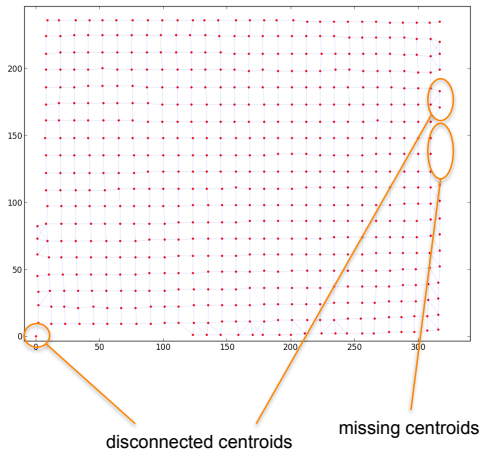


Fig. 4: *Noise and artifacts in the decoding process*

Nowadays many visual encoding solutions exist. QR Codes [9], data matrixes [8] and Microsoft High Capacity Color Barcodes (HCCB) [15] are just a few examples.

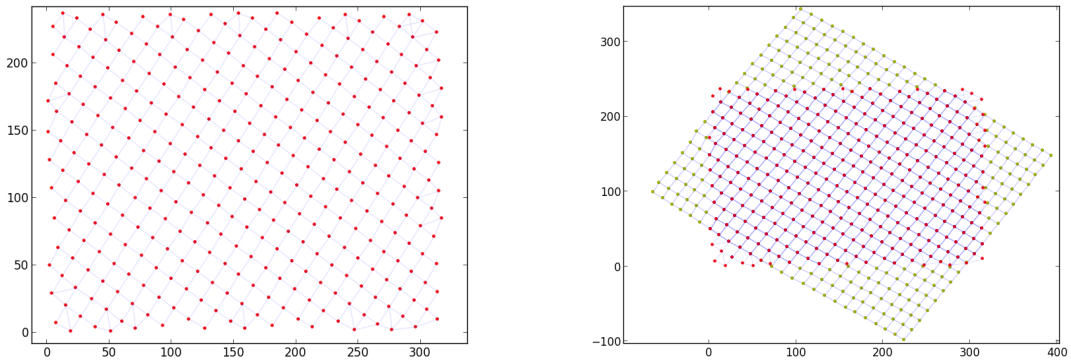
EyeDecrypt is not bound to one specific visual encoding algorithm. It does require, however, that two properties be satisfied.

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.

The second property makes sure that the visual encoding algorithm preserves the spatial ordering of the blocks of ciphertext (which corresponds to the spatial ordering of the plaintext).

Dataglyphs [22] (see Figure 3) 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



(a) Rotated centroids

(b) Rebuilt graph

Fig. 5: *Rebuilding a graph from noisy centroids with unknown rotation*

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. In Section III-B we referred to the dataglyph visual encoding scheme as $\mathcal{V} = (\mathsf{Encode}, \mathsf{Decode})$.

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 $+45°$. An angle of $-45°$ corresponds to the bit 1 and an angle of $+45°$ 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 [7], 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* 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 [10] 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.

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.

Figure 3 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 (ciphertext) image.

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.

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 [**?**] 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.

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 disconnected or missing centroids at the edges of the graph. Figure 4 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 5a 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 5b shows the graph reconstructed from the centroids shown in Figure 5a. We can see that we were able to

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

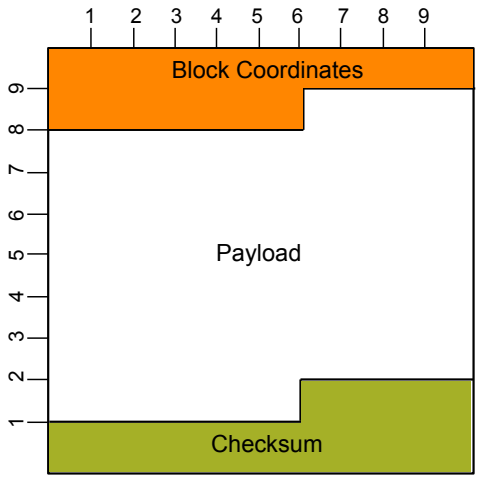


Fig. 6: *Ciphertext block structure*

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 7. 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 6 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 7 shows the correct decoding and decryption of 1,834 bits of ciphertext displayed on a computer screen when holding the phone to about 4 inches away from the display. As we can see, the decoding

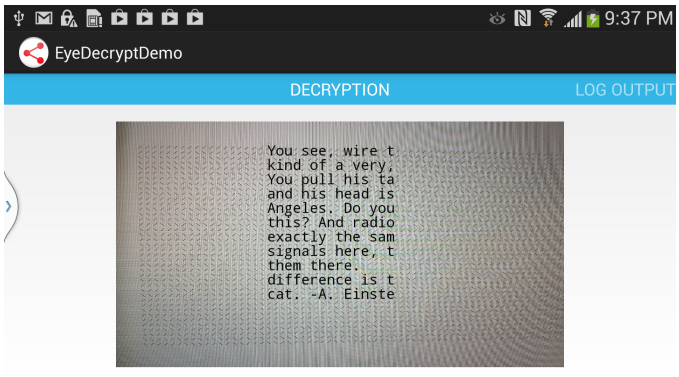


Fig. 7: *Decoding and decryption of 1,834 bits of ciphertext*

was successful despite the presence of Moiré patterns. In such case, the decoding took 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.

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

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 [14], [1], [2]. *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 but we expect it to be much less accessible 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.

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 [13], [19], [20], [4]. They are also used for hiding information (steganography) and for content protection (watermarking).

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. [12], [23], [16]. Techniques for zooming into visual encodings include recursive grids of barcodes [17] and nested barcodes [21]. Fourier tags [18] 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., [3]) but less often in multiple barcodes. Fang and Chang describe a method for encoding a large encrypted message in multiple barcodes [4]. 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.

There are many defenses against shoulder surfing. EyePassword [11] uses gaze-based typing for password entry, in contrast with our augmented reality/randomized keyboard approach. Most other 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* is about making sure that only legitimate users can view the information that is openly displayed, and in that sense bears some similarity to broadcast encryption ([5] 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.

REFERENCES

[1] Michael Backes, Tongbo Chen, Markus Drmuth, Hendrik P. A. Lensch, and Martin Welk. Tempest in a teapot: Compromising reflections revisited. In *IEEE Symposium on Security and Privacy*, pages 315–327, 2009.

[2] Michael Backes, Markus Drmuth, and Dominique Unruh. Compromising reflections-or-how to read LCD monitors around the corner. In *IEEE Symposium on Security and Privacy*, pages 158–169, 2008.

[3] D. Conde-Lagoa, E. Costa-Montenegro, F.J. 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)*, pages 257–258, 2010.

[4] Chengfang Fang and Ee-Chien Chang. Securing interactive sessions using mobile device through visual channel and visual inspection. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 69–78, New York, NY, USA, 2010. ACM.

[5] Amos Fiat and Moni Naor. Broadcast encryption. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 480–491, London, UK, UK, 1994. Springer-Verlag.

[6] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[7] Google. Google glass. http://www.google.com/glass

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

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

[10] itseez. Open Source Computer Vision (OpenCV) Library. http://opencv.org

[11] Manu Kumar, Tal Garfinkel, Dan Boneh, and Terry Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *Proceedings of the 3rd Symposium on Usable Privacy and Security*, SOUPS '07, pages 13–19, New York, NY, USA, 2007. ACM.

[12] Jian Liang, David Doermann, and Huiping Li. Camera-based analysis of text and documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 7(2-3):84–104, July 2005.

[13] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *IEEE Symposium on Security and Privacy*, pages 110–124, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[14] Ko Nishino and Shree K. Nayar. Corneal imaging system: Environment from eyes. *International Journal of Computer Vision*, 70(1):23–40, 2006.

[15] Devi Parikh and Gavin Jancke. Localization and segmentation of a 2D high capacity color barcode. In *IEEE Workshop on Applications of Computer Vision*, pages 1–6. IEEE, 2008.

[16] Samuel David Perli, Nabeel Ahmed, and Dina Katabi. PixNet: interference-free wireless links using LCD-camera pairs. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, pages 137–148, New York, NY, USA, 2010. ACM.

[17] Derek Reilly, Huiqiong Chen, and Greg 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.

[18] 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*, pages 165–174, 2007.

[19] Nitesh Saxena, Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *IEEE Symposium on Security and Privacy*, pages 306–313. IEEE Computer Society, 2006.

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

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

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

[23] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2007.