Automated Side Channel Analysis of Media Software with Manifold Learning

Yuanyuan Yuan, Qi Pang, Shuai Wang*

The Hong Kong University of Science and Technology
{yyuanaq, qpangaa, shuaiw}@cse.ust.hk

Abstract

The prosperous development of cloud computing and machine learning as a service has led to the widespread use of media software to process confidential media data. This paper explores an adversary's ability to launch side channel analyses (SCA) against media software to reconstruct confidential media inputs. Recent advances in representation learning and perceptual learning inspired us to consider the reconstruction of media inputs from side channel traces as a *cross-modality manifold learning* task that can be addressed in a unified manner with an autoencoder framework trained to learn the mapping between media inputs and side channel observations. We further enhance the autoencoder with *attention* to localize the program points that make the primary contribution to SCA, thus automatically pinpointing information-leakage points in media software. We also propose a novel and highly effective defensive technique called *perception blinding* that can perturb media inputs with perception masks and mitigate manifold learning-based SCA.

Our evaluation exploits three popular media software to reconstruct inputs in image, audio, and text formats. We analyze three common side channels — cache bank, cache line, and page tables — and userspace-only cache set accesses logged by standard `Prime+Probe`. Our framework successfully reconstructs high-quality confidential inputs from the assessed media software and automatically pinpoint their vulnerable program points, many of which are unknown to the public. We further show that perception blinding can mitigate manifold learning-based SCA with negligible extra cost.

1 Introduction

Side channel analysis (SCA) infers program secrets by analyzing the target software's influence on physical computational characteristics, such as the execution time, accessed cache units, and power consumption. Practical SCA attacks have been launched on real-world crypto systems [63, 99, 103] to recover crypto keys. With the adoption of cloud computing and machine learning as a service (MLaaS), media software, a type of application software used for processing media files like images and text, is commonly involved in processing private data uploaded to cloud (e.g., for medical diagnosis). Existing works have exploited media software with extensive manual efforts or reconstruct only certain media data [36, 100, 104]. However, the community lacks a systematic and thorough understanding of SCA attack vectors for media software and of the ways that private user inputs of various types (e.g., images or text) can be reconstructed in a unified and automated manner. Hence, this is the first study toward media software of various input formats to assess how their inputs, which represent private user data, can be leaked via SCA in a fully automatic way.

Recent advances in representation learning and perceptual learning [12, 106] inspired us to recast SCA of media software as a cross-modality manifold learning task in which an autoencoder [41] is used to learn the mapping between confidential media inputs and the derived side channel traces in an end-to-end manner. The autoencoder framework can learn a low-dimensional joint manifold of media data and side channel observations to capture a highly expressive representation that is generally immune to noise.

Our proposed autoencoder framework is highly flexible. It converts side channel traces into latent representations with an encoder module ϕ_θ, and the media data in image, audio and text formats can be reconstructed by assembling decoders ψ_θ that correspond to various media data formats to ϕ_θ. By enhancing encoder ϕ_θ with attention [97], the autoencoder framework can automatically localize program points that primarily contribute to the reconstruction of media inputs. That is, the attention mechanism delivers a "bug detector" to locate program points at which information can leak.

The observation that manifold learning captures key perceptions of high-dimensional data in a low-dimensional space [106] inspired us to propose the use of *perception blinding* to mitigate manifold learning–based SCA. Well-designed perception blinding "dominates" the projected low-dimensional perceptions and thus confines adversaries to only

*Corresponding author.

generate media data perceptually bounded to the mask. In contrast, media software that is typically used to process data bytes of media data experiences no extra difficulty in processing the blinded data and recovering the original outputs.

Our evaluation exploits media software, including `libjpeg` [58], `FFmpeg` [1], and `Hunspell` [2], which process media data in image, audio, and text formats. We assess these media software using a common threat model where *trace-based* attackers [15, 28, 45, 91] can log a trace of CPU cache banks, cache lines, or OS page-table entries accessed when executing the media software. We also launch standard `Prime+Probe` attack [88] in userspace-only scenarios and use the logged cache side channels to reconstruct media data. We conduct qualitative and quantitative evaluations of six datasets that represent daily media data whose user privacy can be violated if leaked to adversaries. Our findings show that user inputs can be reconstructed automatically and that the recovered media content, such as images or text, shows considerable (visual) similarity to user inputs. The attention modules facilitate localizing program points that incur input leakage; some have been disclosed before [36, 100, 104], but many, to the best of our knowledge, were previously unknown. Further, we find that perception blinding is highly effective in mitigating manifold learning–based SCA. We also demonstrate the noise resiliency of our attack, and how oblivious RAM [33, 84] can mitigate our attack, though it incurs high cost. In summary, we make the following contributions:

- Advances in cross-modality manifold learning inspired us to advocate SCA of media software as a supervised task that learns a joint manifold of media data and side channel traces. High-quality media data can be reconstructed from side channel traces in a noise-resilient manner without knowledge of the underlying media software implementation or media data formats.
- We enhance autoencoder with attention to localize program points that make notable contributions to information leakage. Furthermore, we design a low-cost perception-blinding technique that effectively mitigates the proposed SCA exploitation.
- Our evaluation subsumes widely used media software used to process images, audio, and text. We demonstrate that high-quality user inputs in various formats can be reconstructed and that perception blinding predominantly impedes our SCA. Our attention-based error-localization technique confirms some program points that have been reported as vulnerable and flags many previously unknown problems in media software.

We released all code and data generated in this research at [3]. We also provide an extended version of this paper at [3] which includes full details of our study.

2 Background

We introduce the high-level procedure of launching SCA in which program inputs are assumed confidential. Let a deterministic and terminating program be P. Executing an input $i \in I$ can be modeled as $P : I \to R$, where R denotes the program behavior during the runtime. Although modern computer architectures prohibit attackers from directly recording R and inferring input $i \in I$, attackers can leverage various *side channels*, which map the runtime behavior of R into an adversarial observation O of certain properties (e.g., cache status) in the execution context of P. The attacker's view can be represented as $view : R \to O$, where given side channel observation O, the attackers leverage composite inverse function $(view \circ P)^{-1} : O \to I$ to map O back to input $i \in I$.

Promising progress has been made by logging (high-resolution) side channels such as accessed cache line, cache bank, or page table entries in an automated manner [21, 36, 63, 99, 100]. Nevertheless, reconstruction of i from logged side channels requires attackers to infer the composite inverse function $(view \circ P)^{-1} : O \to I$. Recovery of such mappings requires an in-depth understanding of how program secrets are propagated (i.e., secret information flow), which could require considerable manual efforts [36, 100] or conducting formal analysis [15, 28, 91]. Note that high-resolution side channels (e.g., cache line access) usually contain millions of records, but only a tiny portion o^* is indeed *input-dependent* [90].

SCA on Media Software. Despite the widespread adoption of MLaaS to process users' private data, the SCA of media software has not been thoroughly examined. For instance, media software is commonly used to process X-ray images because it allows cost-efficient disease diagnosis with cloud resources. However, the leakage of such images on the cloud (e.g., via cache-based side channels [62]) involves a high risk of violating patient privacy. An immense demand exists to gain insights into the extent of privacy problems in media software, given its pervasive use in processing private data. Therefore, we examine real-world media software used to process media data such as photos and daily conversations.

Threat Model and Attack Scenarios. This study reconstructs confidential inputs of media software from side channels. We thus reasonably assume that different inputs of targeted media software can induce distinguishable memory access traces. Otherwise, no information regarding inputs would be leaked.

Profiled SCA [17, 38–40, 45, 66] commonly assumes that side channel logs have been prepared for training and data reconstruction. For our scenario, we generally assume a standard *trace-based* attacker. We assume that a trace of system side channel accesses made by the victim software has been prepared for use. Our evaluated system side channels include cache line, cache bank, and page table entries. The feasibility of logging such fine-grained information has been demonstrated in real-world scenarios [26, 36, 100, 103], and this assumption has been consistently made by many previous works [15, 29, 44, 90, 91, 96]. In this study, we use Intel

Pin [65] to log memory access traces and convert them into corresponding side channel traces (see Sec. 5).

We also benchmark userspace-only scenarios where attackers can launch `Prime+Probe` attack [88] to log cache activities when media software is processing a secret input. We use Mastik [101], a micro-architectural side channel toolkit, to launch "out-of-the-box" `Prime+Probe` and log victim's L1I and L1D cache activities. We pin victim process and `spy` process on the same CPU core; see attack details in Sec. 6.4.

Exploiting new side channels is *not* our focus. We demonstrate our attack over commonly-used side channels. This way, our attack is shown as practically approachable, indicating its high impact and threats under real-world scenarios. Unlike previous SCA on media software [36, 100] or on crypto libraries [105], we do *not* require a "white-box" view (i.e., source code) of victim software. We automatically analyze media software *executables* with different input types. As will be discussed in Sec. 6, we launch manifold learning to reconstruct media data with excellent (visual) similarity to user inputs. Many studies have only flagged program points of information leakage with (unscalable) abstract interpretation or symbolic execution [15, 28, 91]. Direct reconstruction of media data is beyond the scope of such formal method-based techniques, and these studies did not propose SCA mitigation.

3 A Manifold View on SCA of Media Software

This study recasts the SCA of media software as a cross-modality manifold learning task that can be well addressed with supervised learning. We train an autoencoder [41] that maps side channel observations O to the media inputs I of media software. Our threat model (Sec. 2) assumes that attackers can profile the target media software and collect side channel traces derived from many inputs. Therefore, our autoencoder framework is trained to learn from historical data and implicitly forms a low-dimensional joint manifold between the side channel logs and media inputs. We first introduce the concept of manifold, which will help to clarify critical design decisions of our framework (see Sec. 4).

Manifold Learning. The use of manifold underlies the feasibility of dimensionality reduction [54]. The key premise of manifold is the *manifold hypothesis*, which states that real-world data in high-dimensional space are concentrated near a low-dimensional manifold [30]. That is, real-world data often lie in a manifold $\mathcal{M}$ of much lower dimensionality d, which is embedded in its high-dimensional space $\mathcal{R}$ of dimensionality D ($d \ll D$). *Manifold learning* aims to find a projection $f : \mathcal{R} \to \mathcal{M}$ that converts data $x \in \mathcal{R}$ into y in an intrinsic coordinate system of $\mathcal{M}$.[1] f^{-1} projects $f(x) \in \mathcal{M}$ back onto representation x in the high-dimensional space $\mathcal{R}$.

PCA [5] is a linear manifold learning algorithm that aims to find $\mathcal{M}$ by extracting "principal components" of data points [12]. However, most real-world manifolds are nonlinear, and manifold learning algorithms (e.g., ISOMAP) are proposed to project data x onto nonlinear $\mathcal{M}$ [10].

Manifold learning views high-dimensional media data $x \in \mathcal{R}$ as a composite of perceptually meaningful contents that are shown as robust to noise or other input perturbations [31, 106, 107]. Manifold learning algorithms extract expressive representations of high-dimensional data such as images, audio, and text [19, 37], which explains why AI models can make accurate predictions pertaining to high-dimensional data [12]. It is shown that data of the same class (e.g., face photos) generally lie in the same manifold, whereas data of different classes (face vs. vehicle photos) are concentrated on separate manifolds in low-dimensional space [86]. Manifold learning clarifies the inherent difficulty of designing *universal* encoding and generative models applicable to high-dimensional data from different manifolds. The manifold hypothesis has been verified theoretically and empirically [31, 106, 107]. See the extended version [3] for empirical evidence on the validity of manifold hypothesis.

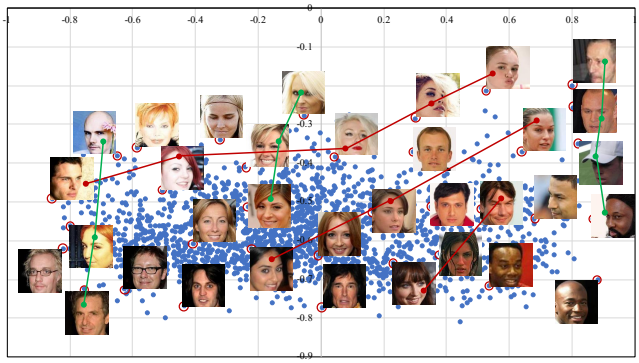

Figure 1: Project face photos to a 2-dimensional manifold.

In Fig. 1, we project a set of real-world face images to manifold $\mathcal{M}_{img}$ of two dimensions. To draw this projection, we tweak our autoencoder framework (see Sec. 4) to convert each face image into a latent representation of two dimensions. We observe that images are generally separated by skin and hair colors; in addition, face orientations are roughly decomposed into two orthogonal directions (green and red lines). In fact, recent studies have shown the effectiveness of conducting image editing by first projecting image samples into the manifold space [83, 107]. Editing images in the high-dimensional space involves a large search space $[0, 255]$ for every pixel; the random selection of pixel values from $[0, 255]$ struggles to create realistic images because arbitrary editing could "fall off" the manifold of natural images. In contrast, manifold learning facilitates sampling within $\mathcal{M}$, and the perceptually meaningful contents in $\mathcal{M}$ confine manipulations to generate mostly realistic images [83, 107].

Parametric Manifold. Most manifold learning schemes adopt non-parametric approaches. Despite the simplicity, non-parametric approaches cannot be used to project *new* data

[1] "Intrinsic coordinate" denotes the coordinate system of the low-dimensional manifold space for each high-dimensional data sample [24, 59].

points in $\mathcal{R}$ onto $\mathcal{M}$. Recent advances in deep neural networks, particularly autoencoders, have enabled a parametric nonlinear manifold projection $f_\theta : \mathcal{R} \to \mathcal{M}$ [106]. Manifold learning can thus process unknown data points of high-dimensional media data [12, 32, 42, 56, 106, 107] and facilitate downstream tasks like face recognition [30].

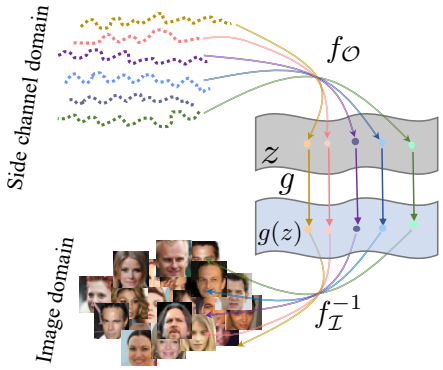

Figure 2: Mapping between side channels and images via a low-dimensional joint manifold $\mathcal{M}_{\mathcal{I},\mathcal{O}} = \mathcal{I} \times \mathcal{O}$.

High-Level Research Overview

Processing media data has an observable influence on the underlying computing environment; it thus induces side channel traces that can be logged by attackers to infer private inputs. Previous SCA studies, from a holistic view, attempted to (manually) map side channel logs to *data bytes* in media data (similar to how media software treats media data) [36, 100]; reconstruction of media data in a per-pixel manner is thus error-prone and likely requires expertise and manual efforts.

The success of manifold learning in tasks like image editing and cross-modality reconstruction [106, 107] led us to construct a joint, *perception-level* connection between side channel logs and high-dimensional media inputs.[2] Therefore, instead of deciding value of each byte, reconstructing media data is recast into exploring the manifold of media data that satisfies the perception-level combinatory constraints.

Overall, we view SCA as a cross-modality high-dimensional data reconstruction task that is addressed with joint manifold learning in this work [106]. Aligned with the notations in Sec. 2, let the media software inputs be I; the attacker's observation on executing each input $i \in I$ can be represented as $(view \circ P) : I \to O$, where $o \in O$ denotes the observation of side channel traces. Let F be the composite function $view \circ P$. According to the manifold hypothesis, we assume that I and O also lie in the unknown manifolds $\mathcal{I}$ and $\mathcal{O}$, respectively. As mentioned in our threat model (Sec. 2), we assume that side channel observations depend on the inputs of media software; therefore, the entire joint dataset $\{i_i, o_i\}$ formed by the ith media input $i_i \in I$ and the corresponding ith observation $o_i \in O$ lies in a joint manifold

$$\mathcal{M}_{\mathcal{I},\mathcal{O}} = \{(i, F(i)) | i \in \mathcal{I}, F(i) \in \mathcal{O}\}$$

where $(i, F(i))$ is described with the regular high-dimensional coordinate system. Since I and O also lie in the corresponding manifolds $\mathcal{I}$ and $\mathcal{O}$, the data points in $\mathcal{M}_{\mathcal{I},\mathcal{O}}$ should be equivalently described using an intrinsic coordinate system $(z, g(z))$. Hence, we assume the existence of a homomorphic mapping $(f_\mathcal{I}, f_\mathcal{O})$ over $(z, g(z))$ such that $z = f_\mathcal{O}(o)$ and $g(z) = f_\mathcal{I} \circ F^{-1}(o)$. $f_\mathcal{O}$ maps side channel observation $\mathcal{O}$ onto the intrinsic coordinate z, whereas $f_\mathcal{I}$ maps high-dimensional media data I onto $g(z)$. Note that g denotes the diffeomorphism (i.e., an isomorphism of two manifolds) between the $\mathcal{I}$ and $\mathcal{O}$ manifolds [106]. Hence, instead of computing $F^{-1} = (view \circ P)^{-1} : O \to I$ to map the side channel observation back onto the media inputs, we leverage the joint manifold to constitute the following composite function:

$$F^{-1}(o) = f_\mathcal{I}^{-1} \circ g \circ f_\mathcal{O}(o) \tag{1}$$

$i \in I$ can thus be reconstructed using the inverse composite function $f_\mathcal{I}^{-1} \circ g \circ f_\mathcal{O}$ over the joint manifold $\mathcal{M}_{\mathcal{I},\mathcal{O}}$. Fig. 2 provides a summary and presents a schematic view of how I and O of high-dimensional data are mapped via $\mathcal{M}_{\mathcal{I},\mathcal{O}}$.

The feasibility of using neural networks, especially autoencoders, to facilitate parametric manifold learning has been discussed [42, 67, 106, 107]. Accordingly, we train an autoencoder by encoding side channel traces O onto the latent space with encoder ϕ_θ and by generating media data I with decoder ψ_θ from the latent space. Therefore, Eq. 1 can be learned in an end-to-end manner [12, 106]. Holistically, ϕ_θ and ψ_θ correspond to $f_\mathcal{O}$ and $f_\mathcal{I}^{-1}$, respectively, whereas g is implicitly constructed in the encoded latent space.

4 Framework Design

We describe the design of our autoencoder in Sec. 4.1. Sec. 4.2 clarifies the usage of attention to localize code fragments inducing information leakage. Sec. 4.3 introduces perception blinding to mitigate our SCA.

4.1 SCA with Autoencoder

We propose a general and highly-flexible design in which an autoencoder is used to facilitate SCA of various media data, including images, audio and text. The autoencoder framework [41] defines a parametric feature-extracting function f_θ, named *encoder*, that enables the projection of the input x onto a latent vector $h = \phi_\theta(x)$. Similarly, autoencoder frameworks also use ψ_θ as a *decoder* that reconstructs input $\hat{x}$ from a latent vector $\hat{x} = \psi_\theta(h)$. A well-trained autoencoder framework gradually identifies a parameter vector θ to minimize the reconstruction error as follows:

$$L(\theta) = \sum_t L(x_t, \psi_\theta \circ \phi_\theta(x_t))$$

[2] Perception-level connection means constraints on data bytes formed by perceptual contents (e.g., gender, hair style) in media data are extracted from side channels.

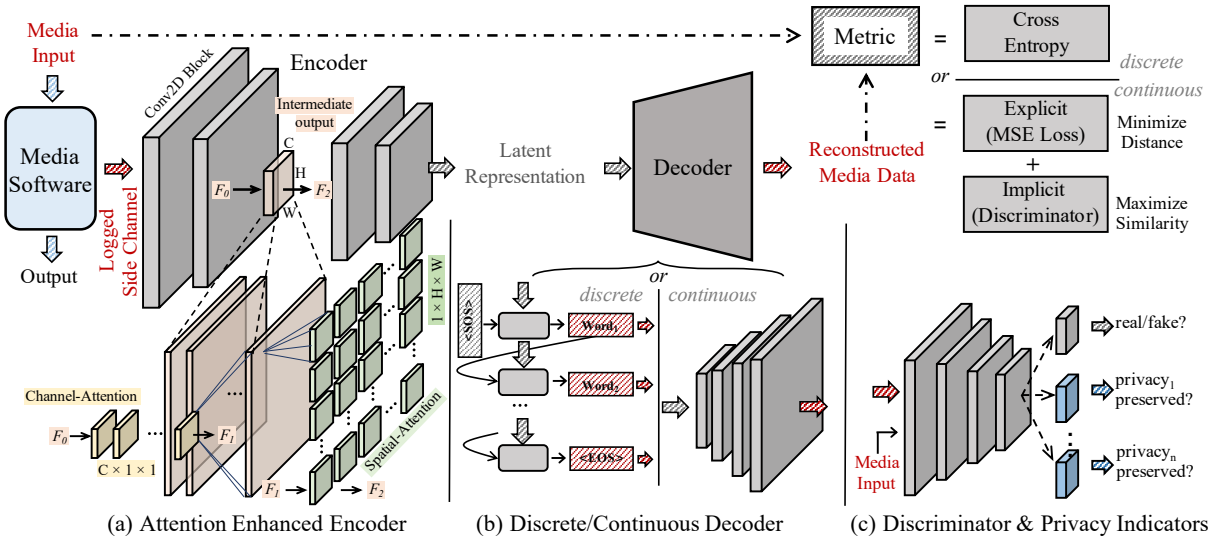

Figure 3: Reconstructing media data of different types with a unified autoencoder framework.

where x_t is a training sample. Minimal errors can be found with statistical methods like stochastic gradient descent.

The first row of Fig. 3 depicts the workflow. We clarify that our focus is *not* to propose novel model architectures; rather, we show that high-quality inputs can be synthesized by assembling standard models, which indicates severity and effectiveness of our attack. We now discuss the high-level workflow and present the model structures and training details in Sec. 5. Given a logged side channel trace $o \in O$, encoder $\phi_\theta(o)$ converts o into the corresponding latent representation. We prepare three decoders ψ^i_θ, ψ^a_θ, ψ^t_θ to reconstruct these types of media data (i.e., image, audio, and text) from the encoded latent representation. We pair encoder $\phi_\theta(o)$ with each ψ^*_θ and train the assembled pipeline for our customized objective functions $L(\theta)$. Our proposed framework is task-agnostic. Generating media data of various types requires only assembling corresponding decoders to the unified encoder ϕ_θ.

Encoder ϕ_θ. A logged side channel trace will first be folded into a $K \times N \times N$ matrix (see Table 3 for the detailed configuration of each trace). We then feed this matrix as the input of encoder ϕ_θ. The encoder ϕ_θ comprises several stacked 2D convolutional neural networks (CNNs). For the current implementation, ϕ_θ converts the high-dimensional inputs into latent vectors of 128 dimensions, given that the dimensions of our media inputs are all over 10K. We visualize encoding of side channel traces, including traces collected from both Intel Pin and `Prime+Probe`, in the extended version of this paper [3]. Moreover, we find that increasing the dimension of latent vectors (i.e., from 128 to 256) does not make an observable improvement. This observation is consistent with the manifold hypothesis [54], such that only *limited* "perceptions" exist in normal media data. In contrast, reducing the number of dimensions (e.g., 32) makes the outputs (visually) much worse. However, users who strive to recover media data of lower-dimensions can configure our framework with smaller latent vectors (e.g., 32 dimensions).

Fig. 2(a) shows that we enhance encoder ϕ_θ with attention. Indeed, we insert one attention module between every two stacked CNN layers in the encoder. Attention generally improves output quality of autoencoder [89]. More importantly, attention facilitates localizing program points of information leakage. We elaborate on Fig. 2(a) in Sec. 4.2.

Decoder ψ^*_θ. We categorize the media data exploited by this study into two types: continuous and discrete. Image and audio data are represented as a continuous floating-point matrix and reconstructed by ψ^i_θ and ψ^a_θ in a continuous manner. In contrast, textual data comprise word sequences, and because there is no "intermediate word," textual data are regarded as sequences of discrete values and handled by ψ^t_θ.

As shown in Fig. 2(b), we use a common approach to stacking 2D CNNs to design ψ^i_θ. A 2D CNN has several convolutional kernels; each kernel focuses on one feature dimension of its input and captures the spatial information of this feature dimension. Images can thus be reconstructed from vectors in the low-dimensional latent space with stacked 2D CNNs, as each 2D CNN upsamples from the output of the previous layer. For audio data, we first convert raw audio into the log-amplitude of Mel spectrum (LMS), a common 2D representation of audio data. This way, audio data are represented as 2D images (see some examples in [3]), in which the x-axis denotes time and the y-axis denotes the log scale of amplitudes at different frequencies. Herein, like ψ^i_θ, ψ^a_θ uses stacked 2D CNNs to process each converted 2D image, gradually upsamples from the latent representation, and reconstruct the LMSs of the audio data. Because the LMSs usually are not in square-shape, we append a fully connected layer to transform the shape of the reconstructed LMSs. These LMSs are then converted to raw audio losslessly.

Textual data, however, are reconstructed sequentially "word by word" due to their discrete nature. As shown in Fig. 2(b), to reconstruct a sentence from the latent space of a side channel trace o, a single word is gradually inferred based on words already inferred from sentence i. Following a common practice of training sequence-to-sequence autoencoders, we add a start-of-sequence (SOS) token before each sentence i and an end-of-sequence (EOS) token after i. Then, given a side channel trace o that corresponds to unknown text i, the trained decoder ψ_θ^t starts from the SOS token and predicts a word $w \in i$ sequentially until it yields the EOS token. From a holistic perspective, the trained model projects a sentence i into a low-dimensional manifold space of *word dependency*, which facilitates the gradual inference of each word w on i.

Designing Objective Functions. As depicted in the first row of Fig. 3, for discrete data (i.e., text), each decode step is a multi-class classification task where the output is classified as one element in a pre-defined dictionary. Thus, we use *cross entropy* as the training objective. For continuous data, we design the training objective L_θ *composing both explicit and implicit metrics*. We now introduce each component in detail.

Explicit Metrics A common practice in training an autoencoder is to explicitly assess the point-wise distance between the reconstructed media input i' and reference input i with metrics such as MSE loss, L1 loss, and KL divergence [20,50]. The autoencoder will be guided to gradually minimize the point-wise distance $L_\theta(i, \psi_\theta \circ \phi_\theta(o))$ during training. Nevertheless, a major drawback of such explicit metrics is that the loss of each data point is calculated *independently* and contributes *equally* to update θ and minimize L_θ. Our preliminary study [3] shows that such explicit metrics suffer from "over-smoothing" [78], a well-known problem that leads to quality degradation of the reconstructed data.

Implicit Metrics Another popular approach is to assess the "distributed similarity" [78] of reconstructed i' and reference input i. Viewing the general difficulty of extracting the distribution of arbitrary media data, a common practice is to leverage a neural discriminator D. Discriminator D and decoder ψ_θ play a zero-sum game, in which D aims to distinguish the reconstructed input i' from normal media data i. In contrast, decoder ψ_θ tries to make its output i' indistinguishable with i to fool D. Although this paradigm generally alleviates the obstacle of "over-smoothing" [78], it creates the new challenge of mode collapse; that is, ψ_θ generates realistic (albeit very limited) i' from any inference inputs. From a holistic perspective, the use of a discriminator mainly ensures that the reconstructed i' is near i from a *distribution* perspective; no guarantee is provided from the view of a single data point.

Privacy-Aware Indicators In addition to the two standard objective functions mentioned above, we further take into account a set of *privacy-aware indicators*. As shown in Fig. 2(c), we extend discriminator D such that it checks whether the reconstructed outputs preserve the "privacy" in an explicit manner. Table 1 lists the privacy indicators used in our framework, which correspond to exploited media data of different types. For instance, for face photos (CelebA), we specify checking the identity. Hence, the enhanced discriminator D serves as a classifier to check whether the identity of the person is preserved, which thus forces decoder ψ_θ to decode the identity information in the zero-sum game. A specific fully connected layer is appended to the discriminator D in accordance with each privacy indicator.

Table 1: Privacy-aware indicators. Table 3 introduces each dataset.

Dataset	Indicator
CelebA	Is the celebrity's identity preserved?
ChestX-ray	Is the disease information preserved?
SC09	Is the speaker's identity preserved?
Sub-URMP	Is the musical instrument's type preserved?

Comparison with Generative Model-Based SCA [104]. One contemporary study [104] uses generative models (e.g., GANs [35]) to conduct SCA towards image libraries by capturing image distribution from side channels. Nevertheless, their work is particularly designed to recover images instead of proposing a general and flexible framework to exploit media software of various input types. In addition, we explicitly use privacy indicators when designing objective functions, while [104] focuses on polishing the *visual appearance* of the reconstructed images.

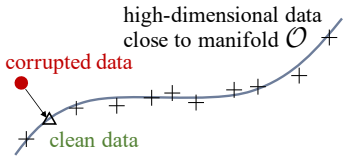

Figure 4: Denoising corrupted data during manifold learning.

Noisy Side Channel. Reconstructing media data from noisy side channels is of particular importance, because adversaries often face considerable noise in real-world attack scenarios. Manifold learning features denoising by design, the schematic view of which is presented in Fig. 4. Overall, manifold learning forces side channel traces O to concentrate near the learned low-dimensional manifold $\mathcal{O}$, where a corrupted high-dimensional data point $\tilde{o}$ (● in Fig. 4) should typically remain *orthogonal* to the manifold $\mathcal{O}$ [12]. Thus, when the decoder ψ_θ learns to reconstruct media data $i \in I$ from the representations lying on the joint manifold, corrupted $\tilde{o}$ can be fixed by first being projected onto the Δ in the manifold for denoising; and i can then be reconstructed from the Δ [42].

4.2 Fault Localization with Neural Attention

Some studies have detected software vulnerabilities that lead to side channel attacks [15, 28, 90, 91]. However, we note that such studies typically use heavyweight program-analysis techniques, such as abstract interpretation, symbolic execution, and constraint solving. Thus, performing scalable program analysis of real-world media software could prove a great

challenge, given that such media software usually contains complex program structures (e.g., nested loops) and a large code base. Furthermore, the primary focus of previous studies has been crypto libraries (e.g., OpenSSL [71]), whose "sensitive data" are private key bytes or random numbers. In contrast, modeling potentially lengthy media data with various strictly defined formats could impose a further challenge (e.g., symbolizing such complex input formats) that may require the incorporation of domain-specific knowledge.

Inspired by advances in program neural smoothing [81, 82] and SCA based on neural networks [45, 74, 104], we seek to overcome question "which program point leaks side channel information" by answering the following question:

> "Which records on a logged side channel trace contribute most to the reconstruction of media data?"

Although answering the former question often requires rigorous and unscalable static analysis, the second question can be addressed smoothly by extending the encoder ϕ_θ with *attention* [97], a well-established mechanism that improves the representation of interest by telling the neural network where and upon what to focus. In particular, by enhancing the autoencoder with attention, our framework *automatically* flags side channel logs that make a primary contribution to input reconstruction. These logs are *automatically* mapped to the corresponding memory access instructions. We can then *manually* identify the corresponding "buggy" source code. For the last step, our current experiments rely on symbol information in the assembly programs to first identify corresponding functions in source code, and then narrow down to code fragments inducing input leakage.

Despite attention being a standard mechanism to boost deep learning models [89, 97], attention in our new scenario acts like a "bug detector" to principally ease localizing vulnerable program points. In contrast to program analysis-based approaches [15, 28, 90, 91], our solution is highly scalable and incurs no extra cost during exploitation. Moreover, it analyzes software in a black-box setting that is agnostic to media software implementation details or input formats.

Fig. 2(a) depicts the enhanced trace encoder with attention. An attention module (we follow the design in [97] given its simplicity and efficiency) is inserted within every two stacked CNN layers. Let the intermediate input of a CNN layer as $C \times H \times W$, the "Channel-Attention" module $A_{channel}$ processes each segment of $1 \times H \times W$ data points from C channels and tells the encoder "where" to focus on by assigning different weights to each segment. The "Spatial-Attention" module $A_{spatial}$ processes each segment of $C \times 1 \times 1$ records and advises the encoder "what to locate" by assigning different weight on each record. From a holistic perspective, attention module $A_{channel}$ projects a coarse-grained focus on potentially interesting segments, while $A_{spatial}$ further identifies interesting side channel records in a segment.[3]

4.3 Mitigation with Perception Blinding

This section presents mitigation against manifold learning–enabled SCA (Sec. 4.1). Consistent with our attack and fault-localization, mitigation is also agnostic to particular media software and input types. We only need to perturb the media input I with pre-defined perception blinding masks.

We first introduce blinding images of the human face, and then explain how to extend perception blinding toward other input types. We also refer readers to our extended paper [3] for visualization of the perception blinding workflow and further clarification on the application scope.

A Working Example. As introduced in Sec. 2, manifold learning casts images of the human face into a set of perceptually meaningful representations; typical representations include hair style, age, and skin color. Hence, we define a *universal* mask i_{mask} of human face, such that by perturbing arbitrary images i of human face with i_{mask}, the produced output $i_{blinded}$ will be primarily projected to the same intrinsic coordinates z_{mask} in the manifold space $\mathcal{M}$. To use perception blinding, users only need to pick *one* mask i_{mask} to blind all input images i. Consequently, adversaries are restricted to the generation of media data perceptually correlated to z_{mask}. Particularly, to perturb i, we add i_{mask} as follows:

$$i_{blinded} = \alpha \times i \oplus \beta \times i_{mask}$$

where we require $\beta \gg \alpha$ and $\alpha + \beta = 1$. Perceptual contents of i_{mask} thus "dominates" the projected low-dimensional perceptions in $\mathcal{M}$. Let $P(i_{blinded})$ be the output of media software after processing $i_{blinded}$, and the user can recover the desired output by subtracting $P(i_{mask})$ from the output as follows:

$$P(i_{private}) = \frac{1}{\alpha} \times (P(i_{blinded}) \ominus \beta \times P(i_{mask}))$$

where $P(i_{private})$ is the desired output, and $P(i_{mask})$ can be precomputed. $\oplus$ and $\ominus$ directly operate $i \in I$ of various formats, as will be defined later in this section. Because typical operations of media software (e.g., compression) are *independent* of the perceptual meaning of media inputs, the proposed blinding scheme introduces no extra hurdle for media software. In contrast, as shown in Sec. 6.3, SCA based on manifold learning can be mitigated in a highly effective manner.

Requirement of i_{mask}. Comparable to how RSA blinding is used to mitigate timing channels [16], perception blinding is specifically designed to mitigate manifold learning-based SCA. We require that i_{mask} must lie in the same low-dimensional manifold with the private data. Thus, i_{mask} must manifest high **perception correlation** with media software inputs $i_{private} \in I$. This shall generally ensure two properties:

[3] It is well accepted that a CNN is organized in the form of `num_channels` × `width` × `height`. Therefore, we name two attention components as $A_{channel}$ and $A_{spatial}$, which are aligned with the convention.

Table 2: Side channels derived from a memory access made by victim media software using address $addr$.

Side Channel Name	Side Channel Record Calculation
CPU Cache Bank Index [103]	$addr \gg L$ where L, denoting cache bank size, is usually 2 on modern computer architectures.
CPU Cache Line Index	$addr \gg L$ where L, denoting cache line size, is usually 6 on modern computer architectures.
OS Page Table Index [36, 102]	$addr \,\&\, (\sim M)$ where M, denoting `PAGE_MASK`, is usually 4095 on modern computer architectures.

Table 3: Statistics of side channel traces and media software. There is *no* overlapping between training and testing data.

Dataset	Information	Training Split	Testing Split	Trace Length	Matrix Encoding	Media Software
CelebA [64]	Large-scale celebrity face photos	162,770	19,962	$338,123 \pm 14,264$	$6 \times 256 \times 256$	`libjpeg` (ver. 2.0.6)
ChestX-ray [92]	Hospital-scale chest X-ray images	86,524	25,596	$329,155 \pm 10,186$	$6 \times 256 \times 256$	LOC: 103,273
SC09 [94]	Human voice of saying number 0–9	18,620	2,552	$1,835,067 \pm 103,328$	$8 \times 512 \times 512$	`ffmpeg` (ver. 4.3)
Sub-URMP [55]	Sound clips of 13 instruments	71,230	9,575	$1,678,485 \pm 36,122$	$8 \times 512 \times 512$	LOC: 1,236,079
COCO [60]	Image captions	414,113	202,654	$77,796 \pm 14$	$6 \times 128 \times 128$	`hunspell` (vers. 1.7.0)
DailyDialog [57]	Sentences of daily chats	11,118	1,000	$77,799 \pm 102$	$6 \times 128 \times 128$	LOC: 39,096

1) the privacy (in terms of certain perceptions, such as gender and skin color) in $i_{private}$ can be successfully "covered" by i_{mask}, and 2) i_{mask} imposes nearly no information loss on recovering $P(i_{private})$ from $P(i_{blinded})$ except a mild computational cost due to mask operations. Considering Fig. 4, when violating this requirement of *perception correlation*, for instance, such as by using random noise to craft i_{mask}, the intrinsic coordinate of the original input (Δ) can likely drift to a "corrupted input" ($\bullet$) that is mostly orthogonal to the manifold of I. As explained in Sec. 4.1, due to the inherent noise resilience of manifold learning, crafting such a corrupted input can cause less challenge to attackers when recovering i from the low-level manifold space. Although $i_{private}$ is of low weight in $i_{blinded}$, it can still be reconstructed to some extent, as will be shown in Fig. 8 of Sec. 6.3.

Extension to other data types. For image and audio data, we recommend using a normal image $i \in I$ as the mask i_{mask}. Intuitively, by amplifying i_{mask} with a large coefficient β in generating $i_{blinded}$, i_{mask} is presumed to dominate the perceptual features in $i_{private}$. Hence, we stealthily hide the private perceptual features of $i_{private}$ in $i_{blinded}$, whose contents are difficult for adversaries to disentangle without knowing i_{mask}. For textual data, we recommend inserting notional words of high frequency to blind $i_{private}$. We present empirical results on how various choices of i_{mask} can influence the mitigation effectiveness in Sec. 6.3.

Implementation of Operators $\oplus$ And $\ominus$. For image and audio data, we use floating-point number addition and subtraction to implement $\oplus$ and $\ominus$. Textual data are discrete: considering that media software often manipulates textual data at the word level, simply "adding" or altering words in the input text will likely trigger some error handling routines of the corresponding media software, which is not desirable. Sec. 4.1 clarifies that our autoencoder framework essentially captures the "word dependency" between words in a sentence; accordingly, we define the $\oplus$ operation as inserting words in a sentence, whereas the $\ominus$ operation is implemented to remove previously inserted words. As shown in Sec. 6.3, this strategy effectively breaks the word dependency in the original text.

5 Attack Setup

We leverage three high-resolution side channels, as shown in Table 2. As clarified in our threat model (Sec. 2), these side channel are commonly adopted in many existing works in this field. We clarify that exploiting new side channels is *not* our focus. We use common side channels in the era of cloud computing, implying the severity and effectiveness of our proposed attack. We refer readers to our extended paper [3] for full details of these side channels. The resolution when performing attacks on those side channels are 4B, 64B, and 4096B, respectively. Higher-resolution side channels should enable recovering media data with more vivid details. Media data of better quality, however, does not necessarily enhance privacy stealing (e.g., determining whether chest X-Ray images indicate pneumonia). See quantitative evaluation of privacy inference in Sec. 6.1.2.

For evaluation in Sec. 6, we use `Pin` [65] to collect memory access traces and map each trace into three side channel traces following mapping rules in Table 2. Sec. 6.4 further demonstrates attack in an userspace-only scenario, i.e., we collect cache side channels via `Prime+Probe` [88].

Media Software and Media Dataset. Table 3 reports evaluated media software and statistics of side channel traces. We pick media software consistent with previous works [36, 100, 104]. All media software are complex real-world software, e.g., `FFmpeg` contains 1M LOC. We prepare two common datasets for each media software to comprehensively evaluate our attack. All datasets contain daily media data that, once exposed to adversaries, would result in privacy leakage. We compile all three media software into 64-bit binary code using `gcc` on a 64-bit Ubuntu 18.04 machine.

Implementation

We implement our framework in PyTorch (ver. 1.4.0). We use the Adam optimizer with learning rate as 0.0002 for all models. Batch size is 64. For continuous decoders, we set the loss function as $\lambda L_{exlicit} + L_{implicit} + \sum_{i=1}^{n} L_{privacy}$, where $\lambda = 50$ and n is the number of privacy-aware indicators. We ran experiments on Intel Xeon CPU E5-2683 with 256 GB

Table 4: CelebA face image matching evaluation.

	Cache bank	Cache line	Page table
same face	45.4%	43.5%	44.5%
non-face	2.0%	2.0%	2.1%

Table 5: SC09 human voice matching evaluation.

	Cache bank	Cache line	Page table
ID accuracy	29.1%	28.8%	23.2%
Content accuracy	21.6%	24.2%	22.6%

RAM and one Nvidia GeForce RTX 2080 GPU. For experiments based on Pin-logged traces (Table 3), the training is completed at 100 epochs and takes less than 24 hours. For experiments using `Prime+Probe`-logged traces, training takes 9 to 27 hours (after excluding preprocessing time; see details in [3]). Table 3 reports the dataset size and training/testing splits. See our released codebase [3] for result verification.

6 Evaluation

We present the SCA exploitation toward media software in Sec. 6.1. We discuss program points that induce information leakage in Sec. 6.2, and demonstrate the effectiveness of perception blinding in Sec. 6.3.

6.1 Side Channel Attack

This section reports key evaluation results. Due to the limited space, full setups and evaluation results are listed in the extended version of this paper [3].

6.1.1 Qualitative Evaluation

This section presents and compares the reconstructed media data with the reference inputs in terms of various settings. Fig. 6 demonstrates that the reconstructed images and the references show highly aligned visual appearances, including gender, eyebrow shapes, skin color, and hair styles. Images constructed from different side channels manifest comparable visual quality. Fig. 5 further reports the text reconstruction results of daily dialogs by comparison with the reference inputs. The reconstructed sentences, although are not fully aligned with the reference, still retain considerable correct contents and the original intents.

We interpret the overall qualitative evaluation results, in terms of images and text, as highly encouraging. We present reconstructed chest X-ray images, sub-URMP/SC09 audio data, and COCO text at [3]. Encouraging results can be consistently observed.

6.1.2 Quantitative Evaluation

Image Data. For CelebA, we leverage commercial face recognition APIs, Face++ [4], to decide whether a reconstructed face and its reference input can be considered as from the *same person* with over 99.9% confidence scores. We thus launch a de-anonymization attack of user identity with reconstructed images. Table 4 reports the evaluation results; for all three exploited side channels, more than 43% of the reconstructed faces can be correctly matched to their reference inputs, showing a high success rate of face matching. Only 2% of the reconstructed images are deemed as "non-face," which indicates the negligible chance of generating corrupted faces. Due to the limited space, we report the quantitative evaluation of chest X-ray in [3].

Our attack achieves plausible accuracy. The quantitative results are *not* noticeably affected by differences in side channels, which indicates that face matching evaluation extracts representative attributes from images for matching. As mentioned in Sec. 5, the three side channels manifest different resolutions: although higher-resolution side channels enable reconstruction of more vivid images, this does not necessarily promote privacy stealing. However, enabled by manifold learning-based autoencoder and our objective functions which explicitly account for privacy indicators (Table 1), privacy-related factors are extracted in the reconstructed images across side channels of various resolutions. Similar observations are made for media data of other formats; our discussion follows.

Audio Data. Table 5 reports the voice matching results for SC09. Using the reconstructed voice commands (number 0–9), we train two classifiers for speaker identity and command 0–9 classification.[4] The evaluation results largely outperform the baseline (i.e., random guessing). With a total of 184 speakers, we achieve greater than 20% accuracy in matching correct speaker identities across all settings. We also exceed 20% accuracy in content matching (0–9). We observed decreasing accuracy in speaker identity matching, which is reasonable given that the cache bank side channel only "kicks off" two least significant bits, while cache line and page table side channels retain less amount of information. We also report the matching rate of musical instruments in [3], which yields consistent and promising findings.

Text Data. To reconstruct text data, we gradually predict each word based on previously predicted words in the sentence. Hence, for the quantitative evaluation, we adopt an attack strategy mostly aligned with [18] to measure the average accuracy of word-level prediction accuracy. Table 6 reports the evaluation results for the COCO and Dailydialog datasets. To prepare a baseline for comparison, we feed a random input to the decoder ψ_θ instead of using the latent vector of an input side channel trace. As expected, our exploitation of both datasets achieves much greater accuracy than the baseline regarding all side channels.

Table 6: Text data inference evaluation.

Dataset	Cache bank	Cache line	Page table	Baseline
COCO Caption	43.4%	42.6%	42.1%	0.0000%
Daily Diolgue	38.1%	37.4%	37.6%	0.0183%

[4] Please refer to [3] for details of these classifiers.

Reconstructed Text	Reference Input
*I think it **' be better for find** a good babysitter here . It **' be cost , an** or three days .*	*I think it would be better to have a good babysitter here . It might even be for two or three days .*
*She **is** a **single** cold , and **it** don ' t want to take **care to** us . But we don ' t like **how** can stay with our .*	*She has a bad cold , and we don ' t want to take her with us . But we don ' t know who can stay with her .*
*I ' m sorry **, say that** . What ' s wrong with her ?*	*I ' m sorry to hear it . What ' s wrong with her ?*

Figure 5: Qualitative evaluation of DailyDialog. We mark ***inconsistent reconstructions***.

Figure 6: Qualitative evaluation of CelebA.

```
1  static void idct32(int *coeffs, int col_limit) {
2    int limit = min(H, col_limit + 4);
3    for (int i = 0; i < H; i++)
4      TR_32(src, src, H, H, limit);
5  }
6  static void TR_32(int *dst, int *src, int dstep,
7                    int sstep, int end) {
8    int o_32[16] = { 0 };
9    for (int i = 0; i < 16; i++)
10     // loading pre-calculated matrix ''transform''
11     for (int j = 1; j < end; j += 2)
12       o_32[i] += transform[j][i] * src[j * sstep];
13   // TR_16 calls TR_8, and TR_8 calls TR_4.
14   TR_16(e_32, src, 1, 2 * sstep, SET, end/2);
15 }
```

Figure 7: Vulnerable code components in FFmpeg. We mark variables depending on FFmpeg's input in red, and **bold** input-dependent memory accesses (line 12).

6.2 Program Point Localization

We present a representative buggy code fragment of FFmpeg in Fig. 7. We present representative buggy code localized in libjpeg and Hunspell in [3]. We also list **all** localized program points in term of assembly code in [3].

libjpeg. We analyze 2,000 media inputs from the CelebA and Chest X-ray datasets. Table 7 reports the localization results of libjpeg. For instance, we identify 7,060 side channel points from 4,000 traces, which can be mapped back to two functions (encode_mcu_huff and decode_mcu) performing minimum coded unit (MCU)-related operations. Similarly, we find information leakage points in modules related to decompression, IDCT, and also output dumping.

Table 3 shows that *one* trace has approximately 400K data points. In other words, Table 7 reveals that a tiny portion of "informative" points on a side channel trace make a primary contribution to information reconstruction. Given an image compressed in JPEG, libjpeg decompresses the image into a bitmap. It is pointed out that the decompression process introduces side channels. IDCT-related functions that were noted by [100] are automatically re-discovered by us. In addition, we identify functions in other image transformation routines (e.g., MCU, upsampling) and output dumping routines that leak inputs. We manually inspected the corresponding implementation of libjpeg and confirmed our findings. Note that our approach is automated and treats the entire libjpeg software as a blackbox, whereas previous studies [36,53,100] could rely on expert knowledge to first localize the vulnerable program points before launching SCA.

Table 7: Localized program points in libjpeg.

Module	#Functions	Frequency	Sample Func. Names
MCU	2	7,060	encode_mcu_huff decode_mcu
Transform	1	5866	jtransform_execute_transform
IDCT	13	4,027	jpeg_idct_15x15 jsimd_idct_ifas
Upsample	15	2,033	h2v1_merged_upsample h2v1_fancy_upsample
Decompress	6	1,352	tjDecompress2 tjDecompressHeader3
Dump	4	8,23	write_bmp_header start_input_bmp

Table 8: Localized program points in FFmpeg.

Module	#Functions	Frequency	Sample Func. Names
Encode	50+	10K+	encode_frame
Decode	50+	10K+	decode_frame
Filter	50+	10K+	filter_frame
IDCT	10+	5K+	idct_idct_32x32_add_ce iadst_idct_16x16_add_c
Dump	10+	5K+	wav_write_trailer wav_write_header

FFmpeg. We use 2,000 inputs from SC09 and Sub-URMP as the inputs of FFmpeg. Our findings, as reported in Table 8, can be mapped to five modules of FFmpeg, each of which contains many functions. FFmpeg processes audio inputs with audio sampling (the sampling frequency is set as 4,000 in our experiments). Fig. 7 presents a vulnerable program component in the IDCT module of FFmpeg where the input "taints" certain variables (marked in red) and eventually influences the memory accesses at line 12. Given different input values, different memory cells are visited at line 12, resulting in access to different cache units or page table entries. In addition, TR_32, TR_8 and TR_4 also suffer from similar patterns (line 14). To the best of our knowledge, side channel issues on FFmpeg are rarely studied, but the findings in FFmpeg are conceptually similar to other media software; for example,

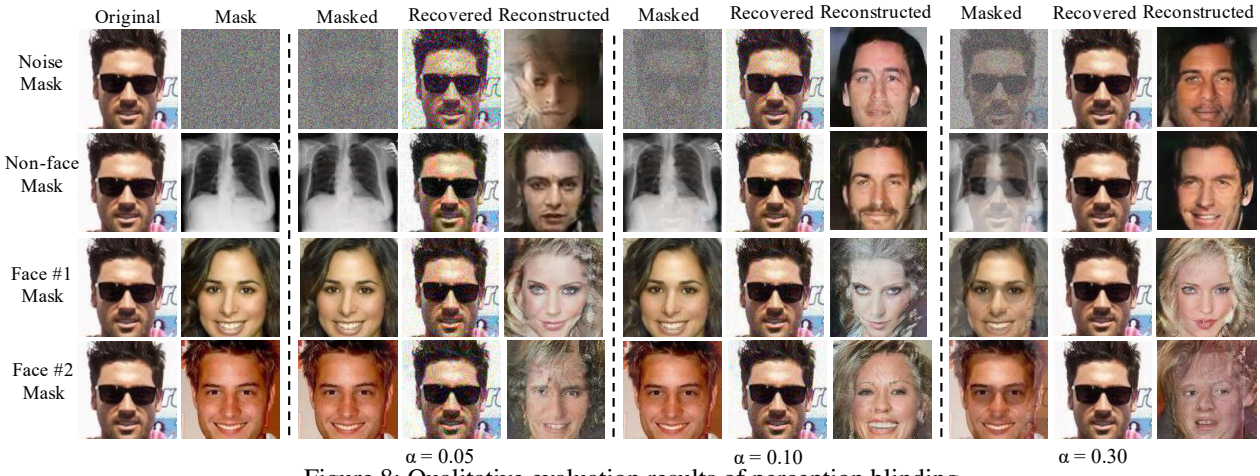

Figure 8: Qualitative evaluation results of perception blinding.

Table 9: Localized program points in `Hunspell`.

Module	#Functions	Frequency	Sample Func. Names
Interface	1	1,333	`pipe_interface`
Parser	4	1,230	`next_token`, `alloc_token` `get_parser`, `TextParser::init`
Look up	2	213	`check`, `insertion_sort`
Insert	5	1,076	`putdic`, `allocate_string` `chenc`, `allocate_char_vector`

IDCT algorithms and output dumping in both `FFmpeg` and `libjpeg` are flagged as vulnerable by us.

`Hunspell`. We use 2,000 inputs from each dataset (i.e., DailyDialog and COCO) to run `Hunspell` and analyze the logged side channels. Table 9 reports the results, where information leakage points are found from the interface, parser, and also spell checking . In fact, previous works [53, 100] have pointed out side channel issues of `Hunspell`. `Hunspell` performs spell check, where a dictionary of words is maintained as a hash table. `Hunspell` iterates each word w in the input sentence to check if w is in the hash table, thus deciding the correctness of its spelling. When checking each w, `Hunspell` computes the hash value of w and looks up the corresponding hash bucket of words. This would lead to a sequence of memory accesses, which can be potentially used to map back to word w. Note that while previous works attacking `Hunspell` assume the knowledge of the dictionary [53, 100] before attack, such pre-knowledge is *not* needed for our attack. Instead, we use side channel traces and their corresponding sentences fed to `Hunspell` as the training data to implicitly learn a mapping in the low-dimensional joint manifold space. [53, 100] reports that functions `lookup` and `add_word` primarily leak inputs. Our manual confirmation shows that our findings (e.g., `putdic`, `chenc`, `check`, `insertion_sort`) indeed invoke `lookup` and `add_word` functions. We also find that the parser and interface (we use Linux utility `echo` to feed `Hunspell`) of `Hunspell` also influence side channels, both of which are not disclosed by previous works.

Confirmation with the Developers. We have reported our localized program points to the developers. By the time of writing, the FFmpeg developers confirmed our findings. Nevertheless, they mentioned that software-level fixing is undesirable, given the difficulty of writing side channel-free code and the incurred extra performance penalty. From his perspective, OS-level or hardware-level fixing seems more practical.

6.3 Mitigation with Perception Blinding

We benchmark the mitigation effectiveness in terms of quantitative and qualitative analysis. We also discuss how different masks can influence the mitigation.

6.3.1 Qualitative Evaluation

We report qualitative evaluation by comparing the reference inputs with the reconstructed inputs after applying blinding. Due to the limited space, Fig. 8 only reports the perception blinding over a private face image $i_{private}$ in terms of different settings. The original image $i_{private}$ is presented in the "Original" column, and applied perception masks are presented in the "Mask" column. For each masked image $i_{blinded}$, the adversarial recovered images are presented in the "Reconstructed" columns, and the final media software outputs after unblinding are given in the "Recovered" columns.

"Noise mask" (the first row) and "non-face mask" (the second row) do not seem helpful in blinding $i_{private}$ because features such as face orientation are still preserved in the reconstructed images. However, the use of real face images as the mask, as shown in the third and fourth columns, gives promising results to blind key perceptual-level contents like hair color and skin color. Overall, after blinding, the adversary-reconstructed images seem to show a correlation with i_{mask} instead of $i_{private}$. This is intuitive; as clarified in Sec. 4.3, a large coefficient β is assigned to i_{mask} such that the perception contents of i_{mask} largely determine the projected intrinsic coordinate in the manifold. This way, the reconstructed images incline to manifest the perception of i_{mask}.

Additionally, although a small α value (e.g., 0.05) introduces a non-trivial amount of noise in the final output, outputs of much better quality can be recovered when α is set to 0.10

Table 10: Face matching results after blinding in terms of (cache bank/cache line/page table).

Mask	$\alpha = 0.05$	$\alpha = 0.1$	$\alpha = 0.3$
Noise	27.5/28.6/27.8%	25.2/26.9/28.2%	26.6/27.5/29.0%
Non-face	28.8/28.8/26.5%	26.2/27.6/27.4%	28.7/31.4/26.2%
Face#1	1.4/1.2/2.4%	1.8/1.4/2.7%	2.0/1.5/3.1%
Face#2	0.6/1.3/1.6%	0.7/1.7/1.9%	1.2/1.6/2.2%

Table 11: Mitigating COCO text inference attack in terms of (cache bank/cache line/page table). $\alpha = 0.05$, $\alpha = 0.1$, $\alpha = 0.3$ denote each word are appended with 19, 9, and 2 masks, respectively.

Mask	$\alpha = 0.05$	$\alpha = 0.1$	$\alpha = 0.3$
"man"	0.39/0.40/0.36%	0.68/0.69/0.67%	2.46/2.45/2.13%
"sitting"	0.16/0.16/0.22%	0.30/0.30/0.39%	1.36/1.40/1.39%

or even higher. We thus recommend that users adopt a reasonably high α when constituting $i_{blinded}$. More reconstructed cases are given in the extended version of this paper [3].

6.3.2 Quantitative Evaluation

We launch quantitative evaluation following the procedures in Sec. 6.1. The perception blinding of "Noise" and "Non-face" masks, as shown in Table 10, reduces the average success rates of face matching from approximately 44% (Table 4) to 27.4%, but still has non-negligible privacy leakage. Compared with "Face#1" and "Face#2", Fig. 8 shows that images reconstructed from "Noise"- and "Non-face"-blinded images manifest better visual similarity with the reference inputs. In addition, Table 10 reports that "Face#1" and "Face#2" exhibit much better mitigation (less than 3.1% matching rates) in terms of quantitative metrics. We find that the value of α does *not* notably influence the results but is still positively correlated with privacy leakage. Overall, for images, we mask the perception contents using blinding. However, the privacy indicator for this scenario, i.e., celebrity's identity, is *not* simply a linear sum of all perception contents. Overall, identity recognition depends on subtle features of a human face: changing α not necessarily impedes capturing informative features.

Table 11 reports the mitigation results of `Hunspell` using COCO. We use two notional words of high frequency, "man" and "sitting", for blinding. We insert N notional words after each word in an input sentence, where N $(\frac{1}{\alpha} - 1)$ is 19, 9, and 2 given different α. When more notional words are used, we observe a higher decrease in inference accuracy. However, with blinding, the inference accuracy decreases from more than 40.0% (see Table 6) to less than 2.5% (close to baseline; see Table 6) even two notional words are inserted after each normal word. Different from masking images, the privacy of text is assessed by word dependency (introduced in Sec. 6.1.2). α decides #notional words inserted to break word dependency. Therefore, the results changes notably w.r.t. values of α.

In sum, our quantitative evaluation demonstrates the effectiveness of our proposed mitigation despite differences in the media data formats or exploited side channels. See our extended paper [3] for more results; for instance, blinding chest X-ray images can drastically reduce the disease diagnosis F1 score from an average of 0.73 to less than 0.1.

6.4 Real-World Attack with `Prime+Probe`

This section explores collecting cache access traces via a practical cache attack, `Prime+Probe` [88, 105], in *userspace-only scenarios*. To do so, we conduct an end-to-end experiment, by leveraging Mastik [101], a micro-architectural side channel toolkit, to perform `Prime+Probe` and log victim's access toward L1D and L1I cache. We use Linux `taskset` to pin the victim software and the `spy` process on the same CPU core. Due to the limited space, we report full details of this attack in the extended version of this paper [3]. We now report key setup and results in this section.

We assume that attackers know when the victim media software begins and ends to process an unknown input [88]. The `spy` process primes and probes the cache. Technically, there is another "coordinator" process on the same core which computes victim process's cache activities and logs the cache side channels to disk. Nevertheless, according to our observation, this coordinator process has generally consistent cache access patterns, thus its mostly fixed cache access does not interfere with our well-trained autoencoder.

We launch experiments on both Intel Xeon CPU and AMD Ryzen CPU. The thresholds of deciding cache hit and cache miss are 120 CPU cycles on Intel Xeon CPU and 100 CPU cycles on AMD Ryzen CPU. `Prime+Probe` is performed in the following manner:

PRIME: The `spy` process fills all cache sets.

IDLE: The attacker logs the access time of all cache sets for the previous `Prime+Probe` iteration. As a result, the idle phase *interval* equals the duration of performing one file I/O operation. Meanwhile, the cache is utilized by the `victim`.

PROBE: The `spy` process refills all cache sets and times the duration to refill the same cache sets to learn how `victim` accesses cache sets.

For a cache set, the logged cache status flip, from hit to miss, indicates at least one cache access of `victim`. We are thus particularly interested in logging such status flip. We name such cache status flips as "cache activity" in the rest of this paper. Whenever a cache activity is observed, we record the cache activities of the all cache sets into a vector V, whose length equals to the number of cache sets. $V[i] = 1$ indicates there is a cache activity in i-th cache set. In other words, the i-th cache set is accessed at least one time by the `victim`. If no cache activity is observed from any cache set, we omit to generate a new V. See full details of this end-to-end attack in our extended paper [3].

The quantitative evaluation results, as reported in Table 12, are generally encouraging. Attacks toward `libjpeg` and `Hunspell` manifest high accuracy comparable with attacks over `Pin`-logged traces (Sec. 6.1.2). While `Prime+Probe` logs relatively noisier cache side channels, our attack shows

Table 12: Quantitative evaluation results using cache side channels logged by `Prime+Probe`. We also provide the processing time (*ms*) when launching `Prime+Probe` (normal → with `Prime+Probe`).

	`FFmpeg` & SC09 voice matching		`libjpeg` & CelebA face matching/non-face		`Hunspell` & DDialog text matching	
	Intel	**AMD**	**Intel**	**AMD**	**Intel**	**AMD**
L1I Cache	12.6% (36 → 580)	11.6% (10 → 420)	38.0/0.85% (5 → 18)	35.9/1.2% (2 → 22)	33.9% (60 → 130)	33.2% (23 → 60)
L1D Cache	81.8% (36 → 590)	15.7% (10 → 420)	36.9/0.80% (5 → 20)	33.9/0.90% (2 → 24)	32.2% (60 → 130)	31.8% (23 → 60)

promising noise resilience, as trace encoder (and manifold learning by design) is noise resilient. Further, the logged side channels are *sparse*; given only a few records are secret-dependent, noise introduced by `Prime+Probe` and other workloads do not primarily impede our attack.

`FFmpeg` reports high attack accuracy (over 80%) on Intel L1D cache but lower accuracy for other settings. As in Table 13, the logged side channel traces are unstable (and challenging to comprehend), where stddev is about half of the average trace length. To verify the high attack accuracy on Intel L1D cache, we manually checked all the reconstructed 2,552 audio clips (also uploaded at [3] for reference). The reconstructed audio clips on Intel L1D cache manifest high quality. However, while the original audio clips of the same class are produced by different persons (and sound very different), all reconstructed audio clips of the same class sound *indistinguishable*. This indicates that the trained model stealthily simplifies the task of reconstruction into a task of ten class-conditional generation (recall this dataset has "0–9" labels). We then manually checked the collected traces: we find that for this particular case, Intel L1D cache "amplifies" the distance of inter-class traces while reduces the distance of intra-class traces. As a result, intra-class differences are not well learned using training data. However, since our quantitative metrics only check if the reconstructed audio can be classified correctly, the attack accuracy is high, indicating privacy leakage. We use attention (Sec. 4.2) and compared all the localized code components contributing side channels: we report that localized functions are the *same* on Intel/AMD CPUs. However, they manifest different frequencies, which result in this subtle model decaying. We confirm that this stealthy issue *only* occurs for this case. To solve this issue (and therefore reconstruct diverse outputs within the same class), users can opt for more complex models or larger training data, if needed.

Overall, inspired by recent work [95] exploring timing-based microarchitectural side channels, we deem it an interesting future work to benchmark the microarchitectural side channel differences. Our tool can automatically check whether side channels are informative enough to reconstruct secrets, when it largely outperforms the baseline.

We report the slowdown incurred by `Prime+Probe` attack in Table 12. Overall, media software are highly complex, and processing media data can usually induce a large volume of cache accesses. This way, frequent cache misses due to `Prime+Probe` can cause a reasonably high slowdown. Our extended paper [3] gives further discussion regarding this.

Qualitative Evaluation Results. Fig. 9 reports reconstructing CelebA face photos on different CPUs. We interpret the reconstruction results as generally promising: a considerable number of visual perceptions are faithfully retained in the reconstructed images, which include gender, face orientation, skin color, nose shape, and hair styles. There is a bit quality degradation on some images, for instance, facial details (e.g., the appearance of teeth) are not always precisely aligned with reference inputs. However, we emphasize that images reconstructed under four settings all manifest comparable visual quality, indicating high feasibility of applying our framework on the basis of commonly-used side channels. The qualitative results are also consistent with results reported in Table 12 — though the attack is launched on different CPUs and different caches, privacy leakage is steadily notable.

6.5 Mitigation Using ORAM

Besides perception blinding, this section assesses other mitigations. Existing mitigations aim at adding randomness, making it constant, or directly masking inputs. Nearly all of them are particularly designed to protect crypto software [16, 23, 87]. Raccoon [76] proposes general mitigation using software obfuscation; however, its implementation is not available. That said, oblivious RAM (ORAM) [33] conceal memory access sequences of a program by continuously shuffling data as they are accessed. We study whether a representative ORAM, PathOHeap [84], can mitigate our attack. Due to the limited space, we report the key evaluation results in Table 14. PathOHeap takes several hours to process one memory access made by `libjpeg`. We thus focus on two critical functions localized by our framework (see Sec. 6.2), `IDCT` and `MCU`, separately.[5] We measure attack success rates with and w/o first converting memory traces using PathOHeap.

Cache line side channels derived from either `IDCT` or `MCU` are sufficient for attack. Nevertheless, ORAM eliminates information leak: memory access traces, after being processed by PathOHeap, do *not* depend on input images. We report that our autoencoder does not even reach convergence during training, and the reconstructed images (using poorly trained autoencoder) show indistinguishable and meaningless visual appearances. The non-zero result (i.e., 0.2%) is because that many face photos look like an "average" face. In other words, 0.2% implies the *baseline* of face matching.

Comparison. PathOHeap is very costly: while `libjpeg` can process an image within 100ms, PathOHeap takes several

[5] Focusing functions with known information leakage (i.e., `IDCT`) [36, 100] demonstrates a "white-box" attacker using our technique.

Table 13: Statistics (*mean*±*stddev*) and matrix encoding of cache side channel traces logged by `Prime+Probe`. Each trace consists of a 64-element vector sequence. The #training and #test splits in each setting are the same as those shown in Table 3.

	`libjpeg`(×8)		`FFmpeg`(×4)		`Hunspell`(×8)	
	Intel	**AMD**	**Intel**	**AMD**	**Intel**	**AMD**
L1I Cache	2719 ± 745 $1 \times 256 \times 256$	1114 ± 290 $1 \times 256 \times 256$	44315 ± 23374 $8 \times 512 \times 512$	7013 ± 2540 $2 \times 512 \times 512$	4834 ± 1788 $4 \times 256 \times 256$	1865 ± 432 $2 \times 256 \times 256$
L1D Cache	780 ± 137 $1 \times 256 \times 256$	5750 ± 931 $4 \times 256 \times 256$	8837 ± 3050 $2 \times 512 \times 512$	46698 ± 35163 $8 \times 512 \times 512$	3739 ± 164 $4 \times 256 \times 256$	9732 ± 1498 $8 \times 256 \times 256$

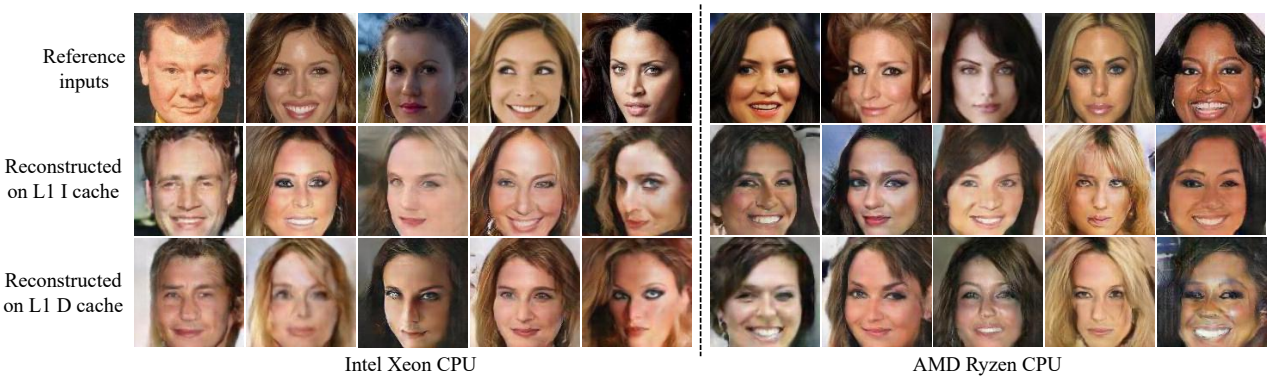

Figure 9: Reconstructed images on L1 cache of Intel Xeon and AMD Ryzen CPU. We can observe highly correlated visual appearances, including gender, face orientation, skin color, nose shape, and hair styles.

Table 14: Attack PathOHeap.

Function	`IDCT`	`MCU`
w/o ORAM	40.3%	38.0%
with ORAM	0.2%	0.2%

Table 15: Quantitative evaluation results (same face/non-face) of face images reconstructed from noisy side channels.

Setting	Noise Insertion Scheme	**NA**	**Low**	**High**
`Pin` logged trace	**Gaussian**	43.5/2.0%	33.8/2.1%	28.0/1.5%
	Shifting	43.5/2.0%	42.9/1.7%	39.1/1.8%
	Removal	43.5/2.0%	30.0/1.9%	29.3/4.3%
`Prime+Probe` logged trace	**Leave out**	36.9/0.8%	36.8/1.1%	36.8/1.1%
	False hit/miss	36.9/0.8%	36.4/1.2%	36.1/1.2%
	Wrong order	36.9/0.8%	36.8/1.0%	36.7/1.0%
Workload under `Prime+Probe`	**Bzip2**	36.9/0.8%	27.6/1.0%	
	Victim$_1$	36.9/0.8%	30.7/1.1%	
	Victim$_2$	36.9/0.8%	29.0/1.0%	

hours to convert the corresponding memory trace. The obfuscator, Racoon [76], has an average overhead of 16.1×. In contrast, perception-blinding delivers negligible extra cost (i.e., processing masked data using media software once) albeit its mitigation is specific for manifold learning-based SCA. This underlines the key novelty of our technique.

6.6 Noise Resilience

We have discussed the general immunity to noise of manifold learning in Fig. 4. This section empirically assesses our attacks under various scenarios where noise is introduced in side channels. We summarize our noise insertion schemes in Table 15. The first three schemes are launched to mutate cache line access traces logged by `Pin`, whereas the latter three mutate the cache set hit/miss records logged via `Prime+Probe`. **NA** means no noise is inserted, whereas **Low/High** denotes to what extent side channel logs are perturbed (see [3] for further details). We also benchmark how real-world workload, i.e., by launching `bzip2` or another victim software (e.g., `libjpeg`) on the same CPU core, can undermine our attack. **Victim$_1$** and **Victim$_2$** represent launching another victim software on the same core and processing the same or different inputs.

Due to the limited space, we only report the quantitative evaluation results of `libjpeg` on CelebA in Table 15. See our extended paper [3] for other quantitative and qualitative results: despite the applied noise, perceptual features are still retained in the reconstructed data (e.g., face photos), illustrating the feasibility of privacy stealing under noisy scenarios.

The reconstructed images are more resilient toward **Shifting**. **Removal** and **Gaussian** noise, by extensively leaving out or perturbing data points on the logged trace (e.g., **Removal/High** removes *half* records on a trace), show greater influence on data reconstruction. As for noise inserted in `Prime+Probe` logged side channel records, none of them primarily affect the attack accuracy. We note that `Prime+Probe` logged side channel traces, even without applying these noise insertion schemes, are of high stddev. That is, our autoencoder will be trained with more "diverse" side channel logs, which enhance the robustness but undermines accuracy. Similar findings are obtained in launching extra real-world workloads. Please refer to the extended version [3] of this paper for further evaluation on noise resilience, and our analysis on noise resilience from the trace encoder structure perspective and training data perspective.

7 Related Work

Side Channel Analysis. Kocher proposes to use timing side channel to exploit crypto systems [46]. To date, side channels have been used to exploit crypto systems under different scenarios [7, 11, 27, 99], including trusted computing environments like Intel SGX [13, 53, 68, 80]. [16] demonstrates that timing side channel can be launched remotely through network. The CPU cache are particularly exploited given its indispensable role in boosting modern computing platforms [36, 63, 72, 103]. Controlled side channel assumes an adversarial-controlled OS to log page table access of victim software [100]. DNNs have been used to infer secret keys from crypto libraries [39, 40, 66]. These works, usually referred to as "profiled SCA", share the same assumption with our research that models are trained using historical data. Most existing DNN-based SCA focuses on attacking crypto systems; they typically perform low-level *bit-wise classification* to gradually infer key bits. In contrast, we show that attackers in black-box scenarios can use manifold learning to reconstruct media data of various types in an end-to-end manner. [51, 98] also use autoencoders in the context of SCA. However, they use autoencoder to denoise side channel traces as a *preprocessing* step for SCA of crypto software.

Countermeasures. Software-based techniques include constant-time techniques which ensure that software behavior is independent with its confidential data [22, 47, 69, 75, 79]. Techniques have also been proposed to blind secrets or randomize side channel access patterns [9, 14, 25, 43, 49, 76]. ORAM [34, 61, 84, 85] translates memory access into identical or indistinguishable traces, which can provably eliminate many side channels but incur high performance penalty. Program analysis methods such as information flow tracking [52, 70], model checking [8], type system [6, 77], abstract interpretation [28, 48, 90], and constraint solving [15, 90, 91] are used to check crypto software and detect side channels. In contrast, our study delivers a *neural attention*-based approach to detecting code fragments inducing information leakage. Hardware-based countermeasures include randomizing side channel access or enforcing fine-grained resource isolation [62, 73, 93]. Compared with system- and hardware-based countermeasures, software-based approaches usually do not require to modify the underlying hardware design. Nevertheless, software-based countermeasures are generally high cost and low scalable in analyzing real-world software.

8 Conclusion

This research proposes SCA for media software. We perform cross-modality manifold learning to reconstruct media data from side channel traces. We also use attention to localize program points leading information leakage. We design perception blinding to mitigate the proposed SCA. Our evaluation on real-world media software reports promising results.

References

[1] FFMPEG. https://ffmpeg.org/.

[2] hunspell. http://hunspell.github.io/.

[3] Research artifact & extended version. https://github.com/Yuanyuan-Yuan/Manifold-SCA.

[4] Face attributes analysis service. https://www.faceplusplus.com/attributes/, 2020.

[5] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[6] Johan Agat. Transforming out timing leaks. POPL, 2000.

[7] Nadhem J Al Fardan and Kenneth G Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. IEEE Security & Privacy, 2013.

[8] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *USENIX Sec.*, 2016.

[9] Aslan Askarov, Danfeng Zhang, and Andrew C Myers. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 297–307, 2010.

[10] Mukund Balasubramanian, Eric L Schwartz, Joshua B Tenenbaum, Vin de Silva, and John C Langford. The ISOMAP algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

[11] Lucas Bang, Abdulbaki Aydin, Quoc-Sang Phan, Corina S Păsăreanu, and Tevfik Bultan. String analysis for side channels with segmented oracles. FSE, 2016.

[12] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[13] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. WOOT, 2017.

[14] Benjamin A Braun, Suman Jana, and Dan Boneh. Robust and efficient elimination of cache and timing side channels. *arXiv preprint arXiv:1506.00189*, 2015.

[15] Robert Brotzman, Shen Liu, Danfeng Zhang, Gang Tan, and Mahmut Kandemir. CaSym: Cache aware symbolic execution for side channel detection and mitigation. In *IEEE SP*, 2018.

[16] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *CHES*, 2017.

[18] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. USENIX Security, 2019.

[19] Ya Chang, Changbo Hu, and Matthew Turk. Manifold of facial expression. In *AMFG*, pages 28–35, 2003.

[20] Mikhail Chernov and Eric Ghysels. A study towards a unified approach to the joint estimation of objective and risk neutral measures for the purpose of options valuation. *Journal of financial economics*, 56(3):407–458, 2000.

[21] R. C. Chiang, S. Rajasekaran, N. Zhang, and H. H. Huang. Swiper: Exploiting virtual machine vulnerability in third-party clouds with competition for i/o resources. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1732–1742, June 2015.

[22] Bart Coppens, Ingrid Verbauwhede, Koen De Bosschere, and Bjorn De Sutter. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *IEEE SP*, 2009.

[23] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *CHES*, 2009.

[24] Jose A Costa and Alfred O Hero. Geodesic entropic graphs for dimension and entropy estimation in manifold learning. *IEEE Transactions on Signal Processing*, 52(8):2210–2221, 2004.

[25] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. Thwarting cache side-channel attacks through dynamic software diversity. NDSS, 2015.

[26] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen. Prime+Abort: A timer-free high-precision L3 cache attack using Intel TSX. In *USENIX Sec.*, 2017.

[27] Xiaowan Dong, Zhuojia Shen, John Criswell, Alan L Cox, and Sandhya Dwarkadas. Shielding software from privileged side-channel attacks. In *27th USENIX Security Symposium*, pages 1441–1458, 2018.

[28] Goran Doychev, Dominik Feld, Boris Kopf, Laurent Mauborgne, and Jan Reineke. CacheAudit: A tool for the static analysis of cache side channels. In *USENIX Sec.*, 2013.

[29] Goran Doychev and Boris Köpf. Rigorous analysis of software countermeasures against cache attacks. PLDI, 2017.

[30] Xing Fan, Wei Jiang, Hao Luo, and Mengjuan Fei. Spherereid: Deep hypersphere manifold embedding for person re-identification. *Journal of Visual Communication and Image Representation*, 60:51–58, 2019.

[31] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[32] Zhenyong Fu, Tao Xiang, Elyor Kodirov, and Shaogang Gong. Zero-shot object recognition by semantic manifold distance. CVPR, 2015.

[33] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. STOC, 1987.

[34] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[35] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. NIPS, 2014.

[36] Marcus Hähnel, Weidong Cui, and Marcus Peinado. High-resolution side channels for untrusted operating systems. USENIX ATC, 2017.

[37] Xiaofei He, Shuicheng Yan, Yuxiao Hu, Partha Niyogi, and Hong-Jiang Zhang. Face recognition using laplacianfaces. *IEEE transactions on pattern analysis and machine intelligence*, 27(3):328–340, 2005.

[38] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled power analysis attacks using convolutional neural networks with domain knowledge. SAC, 2018.

[39] Benjamin Hettwer, Tobias Horn, Stefan Gehrer, and Tim Güneysu. Encoding power traces as images for efficient side-channel analysis. *arXiv preprint arXiv:2004.11015*, 2020.

[40] Annelie Heuser and Michael Zohner. Intelligent machine homicide. In *COSADE*, 2012.

[41] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, 6:3–10, 1994.

[42] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, pages 1–4. 2015.

[43] Wei-Ming Hu. Reducing timing channels with fuzzy time. *Journal of computer security*, 1(3-4):233–254, 1992.

[44] Gorka Irazoqui, Kai Cong, Xiaofei Guo, Hareesh Khattri, Arun K. Kanuparthi, Thomas Eisenbarth, and Berk Sunar. Did we learn from LLC side channel attacks? A cache leakage detection tool for crypto libraries. *CoRR*, 2017.

[45] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[46] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. CRYPTO, 1996.

[47] Boris Köpf and Heiko Mantel. Transformational typing and unification for automatically correcting insecure programs. *International Journal of Information Security*, 6(2-3):107–131, 2007.

[48] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In *CAV*, 2012.

[49] Boris Köpf and Geoffrey Smith. Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 44–56. IEEE, 2010.

[50] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[51] Donggeun Kwon, HeeSeok Kim, and Seokhie Hong. Improving non-profiled side-channel attacks using autoencoder based preprocessing. *IACR Cryptol. ePrint Arch.*, 2020:396, 2020.

[52] Adam Langley. ctgrind. https://github.com/agl/ctgrind.

[53] Dayeol Lee, Dongha Jung, Ian T Fang, Chia-Che Tsai, and Raluca Ada Popa. An off-chip attack on hardware enclaves via the memory bus. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[54] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[55] Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications. *IEEE Transactions on Multimedia*, 21(2):522–535, 2018.

[56] Haoqi Li, Brian Baucom, and Panayiotis Georgiou. Unsupervised latent behavior manifold learning from acoustic features: Audio2behavior. ICASSP, 2017.

[57] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957*, 2017.

[58] libjpeg. Main libjpeg-turbo repository, 2020.

[59] Tong Lin and Hongbin Zha. Riemannian manifold learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):796–809, 2008.

[60] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[61] Chang Liu, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, and Elaine Shi. Ghostrider: A hardware-software system for memory trace oblivious computation. *ACM SIGPLAN Notices*, 50(4):87–101, 2015.

[62] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *HPCA*, 2016.

[63] Fangfei Liu, Y. Yarom, Qian Ge, G. Heiser, and R.B. Lee. Last-level cache side-channel attacks are practical. IEEE SP, 2015.

[64] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. ICCV, 2015.

[65] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI'05)*, 2005.

[66] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[67] Francisco J Martinez-Murcia, Andres Ortiz, Juan-Manuel Gorriz, Javier Ramirez, and Diego Castillo-Barnes. Studying the manifold structure of alzheimer's disease: A deep learning approach using convolutional autoencoders. *J-BHI*, 24(1):17–26, 2019.

[68] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How sgx amplifies the power of cache attacks. In *CHES*, 2017.

[69] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In *ICISC*, 2005.

[70] Andrew C Myers. JFlow: Practical mostly-static information flow control. PLDI, 1999.

[71] Openssl. https://www.openssl.org/.

[72] Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1406–1418, 2015.

[73] D. Page. Partitioned cache architecture as a side-channel defence mechanism, 2005.

[74] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 157–176. Springer, 2018.

[75] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced cloud services. In *CCSW*, 2009.

[76] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 431–446, 2015.

[77] Andrei Sabelfeld and Andrew C Myers. Language-based information-flow security. *IEEE Journal on*

selected areas in communications, 21(1):5–19, 2003.

[78] Yuki Saito, Shinnosuke Takamichi, and Hiroshi Saruwatari. Statistical parametric speech synthesis incorporating generative adversarial networks. *TASLP*, 26(1):84–96, 2017.

[79] Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, and Stefan Mangard. KeyDrown: Eliminating software-based keystroke timing side-channel attacks. In *NDSS*, 2018.

[80] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using sgx to conceal cache attacks. *arXiv preprint arXiv:1702.08719*, 2017.

[81] Dongdong She, Yizheng Chen, Abhishek Shah, Baishakhi Ray, and Suman Jana. Neutaint: Efficient dynamic taint analysis with neural networks. IEEE SP, 2020.

[82] Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, and Suman Jana. NEUZZ: Efficient fuzzing with neural program smoothing. IEEE SP, 2019.

[83] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, pages 9243–9252, 2020.

[84] Elaine Shi. Path oblivious heap: Optimal and practical oblivious priority queue. IEEE SP, 2020.

[85] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious ram protocol. In *CCS*, 2013.

[86] Nicolas Thorstensen. *Manifold learning and applications to shape and image processing*. PhD thesis, Ecole des Ponts ParisTech, 2009.

[87] Kris Tiri and Ingrid Verbauwhede. Securing encryption algorithms against dpa at the logic level: Next generation smart card technology. In *CHES*, 2003.

[88] Eran Tromer, DagArne Osvik, and Adi Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[89] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[90] Shuai Wang, Yuyan Bao, Xiao Liu, Pei Wang, Danfeng Zhang, and Dinghao Wu. Identifying cache-based side channels through secret-augmented abstract interpretation. USENIX Security, 2019.

[91] Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu. CacheD: Identifying cache-based timing channels in production software. In *26th USENIX Security Symposium*, pages 235–252, 2017.

[92] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. CVPR, 2017.

[93] Zhenghong Wang and Ruby B Lee. A novel cache architecture with enhanced performance and security. In *MICRO*, 2008.

[94] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[95] Daniel Weber, Ahmad Ibrahim, Hamed Nemati, Michael Schwarz, and Christian Rossow. Osiris: Automatic Discovery of Microarchitectural Side Channels. In *USENIX Security Symposium*, 2021.

[96] Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar. MicroWalk: A framework for finding side channels in binaries. In *ACSAC*, 2018.

[97] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. ECCV, 2018.

[98] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *TCHES*, pages 389–415, 2020.

[99] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. USENIX Security, 2012.

[100] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. IEEE SP, 2015.

[101] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. *Retrieved from School of Computer Science Adelaide: http://cs.adelaide.edu.au/yval/Mastik*, 16, 2016.

[102] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, pages 719–732, 2014.

[103] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: a timing attack on openssl constant-time rsa. *Journal of Cryptographic Engineering*, 7(2):99–112, 2017.

[104] Yuanyuan Yuan, Shuai Wang, and Junping Zhang. Private image reconstruction from system side channels using generative models. In *ICLR*, 2021.

[105] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. CCS, 2012.

[106] Bo Zhu, Jeremiah Z Liu, Stephen F Cauley, Bruce R Rosen, and Matthew S Rosen. Image reconstruction by domain-transform manifold learning. *Nature*, 555(7697):487–492, 2018.

[107] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, pages 597–613. Springer, 2016.