Preventing Side-Channel Leaks in Web Traffic: A Formal Approach

Michael Backes
Saarland University and MPI-SWS
backes@mpi-sws.org

Goran Doychev
IMDEA Software Institute
goran.doychev@imdea.org

Boris Köpf
IMDEA Software Institute
boris.koepf@imdea.org

Abstract

Internet traffic is exposed to potential eavesdroppers. Standard encryption mechanisms do not provide sufficient protection: Features such as packet sizes and numbers remain visible, opening the door to so-called side-channel attacks against web traffic.

This paper develops a framework for the derivation of formal guarantees against traffic side-channels. We present a model which captures important characteristics of web traffic, and we define measures of security based on quantitative information flow. Leaning on the well-studied properties of these measures, we provide an assembly kit for countermeasures and their security guarantees, and we show that security guarantees are preserved on lower levels of the protocol stack.

We further propose a novel technique for the efficient derivation of security guarantees for web applications. The key novelty of this technique is that it provides guarantees that cover all *execution paths in a web application, i.e. it achieves completeness. We demonstrate the utility of our techniques in two case studies, where we derive formal guarantees for the security of a medium-sized regional-language Wikipedia and an auto-complete input field.*

1 Introduction

Internet traffic is exposed to potential eavesdroppers. To limit disclosure of secret information, security-aware users can protect their traffic by accessing web services which offer TLS encryption, or by sending their data through encrypted tunnels. While today's encryption mechanisms hide the secret payload from unauthorised parties, they cannot hide lower-level traffic features such as packet sizes, numbers, and delays. These features contain information about the payload which can be extracted by traffic profiling, side-stepping the protection offered by cryptography. The relevance of this threat is demonstrated by a large number of side-channel attacks against encrypted web traffic, e.g. [4, 7, 8, 10, 12, 22, 23, 30, 32, 40].

A number of approaches for the mitigation and analysis of side-channel leaks have been proposed, e.g. [30, 33, 40, 41]. A common pattern in these approaches is the following relationship between attacks, countermeasures, and their security analysis: An *attack* corresponds to a classification of a sample of network traffic, where classes correspond to secret aspects of user behavior. A correct classification corresponds to a successful attack, i.e. one in which the secret is correctly recovered. A *countermeasure* modifies the shape of the network traffic with the goal of making the classification of samples more difficult or even impossible. A *security analysis* is based on an evaluation of the performance of a particular classification algorithm.

A security analysis following this pattern enables one to assess a system's vulnerability to a particular classifier; however, the analysis does not make immediate assertions about the vulnerability to attackers using more sophisticated techniques for mining their observations. This limitation is not only unsatisfactory from a theoretical viewpoint, but it also raises significant problems in practice: A recent comprehensive study [17] of traffic analysis countermeasures exhibits that the incompleteness of existing security analyses indeed leaves room for highly effective attacks. It is clear that, ultimately, one strives for security guarantees that hold for *all* realistic adversaries who can observe the web application's traffic, i.e. formally backed-up security guarantees.

There are two key challenges in deriving formal guarantees against side-channel attacks against web traffic. The first challenge is to devise a *mathematical model* of the web application's traffic, which is indispensable for expressing and deriving security guarantees. Such a model has to be accurate enough to encompass all relevant traffic features, and at the same time, the model must be simplistic enough to allow for tractable reasoning about realistic web applications.

The second challenge is to develop *techniques for the computation* of security guarantees for a given web application. The main obstacle is that the derivation of security guarantees requires considering all execution paths of a web application. As the number of paths may grow expo-

nentially with their length, their naive enumeration is computationally infeasible. Previous approaches deal with this problem by resorting to a subset of the possible execution paths [5, 42], introducing incompleteness into the analysis.

In this paper, we develop a novel framework for the derivation of formal guarantees against traffic side-channels in web applications. We first provide a model that captures the effect of user actions on web-traffic. In particular, we cast web applications as labeled, directed graphs where vertices correspond to the states of the web application, edges correspond to possible user actions, and (vertex) labels are observable features of encrypted traffic induced by moving to the corresponding state. These features can include packet sizes and numbers, and their source and destinations, which is sufficient to encompass relevant examples.

We then interpret this graph as an information-theoretic *channel* mapping sequences of (secret) user actions to (public) traffic observations. Casting the graph as a channel allows us to apply established measures of confidentiality (Shannon entropy, min-entropy, g-leakage) from quantitative information-flow analysis and make use of their well-understood properties. Leveraging those properties, we obtain the following results.

First, we provide an assembly kit for countermeasures and their security guarantees. In particular, we identify a number of basic countermeasures, and we demonstrate how representative countermeasures from the literature can be obtained by their composition. The information that is revealed by a composed countermeasure is upper-bounded by the information that is revealed by underlying components (formalized as the so-called data processing inequality), which allows us to show how composed countermeasures relate in terms of the security they provide. Second, we show that the security guarantees derived in our model (which capture attackers who observe encrypted traffic at a specific layer of the protocol stack) also hold for attackers who observe the traffic at lower layers of the stack. We obtain this result by casting segmentation of packets and packet headers in our assembly kit, and by applying the data processing inequality. Third, we show that recent work on predictive mitigation of timing channels [2] can be applied in our setting, allowing us to deal with leaks induced by traffic patterns and timing in a modular fashion.

To put our model into action, we propose a novel technique for the efficient derivation of security guarantees for web applications. The key advantage of this technique is that it allows considering *all* execution paths of the web application, which is fundamental for deriving formal security guarantees. We achieve this result for the important special case of a user following the *random surfer model* [36] and for Shannon entropy as a measure: As a first step, we use PageRank to compute the stationary distribution of a random surfer, which we take as the a priori probability of a user visiting a vertex. As a second step, we apply the chain rule of entropy to reduce the computation of the uncertainty about a path of length ℓ to that about path of length $\ell - 1$, and transitively to that about a single transition. As computing the uncertainty about a single transition can be done efficiently, this is the key to avoiding the enumeration of all (exponentially many) paths.

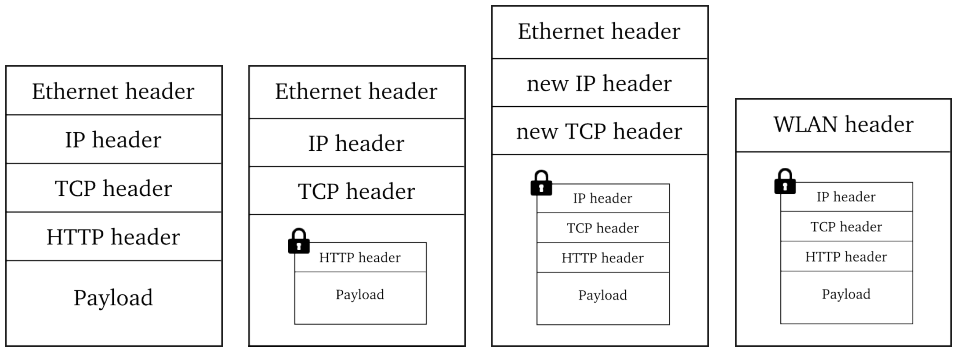
We use this algorithm to study the trade-off between security and performance (in terms of overhead) of different countermeasures. A key observation is that countermeasures based on making individual vertices indistinguishable (e.g. [41]) fail to protect paths of vertices. To ensure protection of paths, we devise *path-aware* countermeasures, which strengthen countermeasures for vertices by making sure that indistinguishable vertices also have indistinguishable sets of successors. Formally, we achieve this by coarsening indistinguishability on vertices to a probabilistic bisimulation relation, which we compute using a novel algorithm based on randomization and partition refinement.

We demonstrate the applicability of the proposed techniques in two case studies, where we analyze the traffic of (1) a regional-language Wikipedia consisting of 5,000 articles, and that of (2) an auto-complete input field for a dictionary of 1,100 medical terms. Our approach delivers a number of countermeasures that provide different trade-offs between security guarantees and traffic overhead, spreading from good security at the price of a high overhead, to low overhead at the price of lower security guarantees.

In summary, our main contributions are twofold. First, we present a novel model for formal reasoning about side-channel attacks in web applications. We show that this model is expressive enough to capture relevant aspects of web applications and countermeasures, and that it provides a clean interface to information theory and quantitative information-flow analysis. Second, we present algorithms and techniques that enable the efficient derivation of such guarantees for real systems and we demonstrate how they can be used for adjusting the trade-off between security and performance.

2 Web-traffic as an information-theoretic channel

In this section, we cast web-traffic as an information-theoretic channel (i.e. a conditional probability distribution) from user input to observable traffic patterns. We specify the threat scenario in Section 2.1 and present our basic model of web applications and their traffic in Section 2.2.



(a) An Ethernet frame without encryption

(b) An Ethernet frame with HTTPS encryption

(c) An Ethernet frame with an encrypted tunnel to a proxy server

(d) A WLAN frame with WPA2 encryption

Figure 1. Encrypted packets with commonly used protection mechanisms. When HTTPS is applied (Figure 1(b)), the attacker may use the information contained in the TCP header to reassemble the packets and infer the approximate size of the web-object. When traffic is forced through an SSH tunnel (Figure 1(c)) and when using an encrypted wireless connection (Figure 1(d)), the original headers are encrypted and direct reassembly is not possible any more.

2.1 Threat scenario

The threat scenario we are considering is a user performing confidential actions (such as following hyperlinks, or typing characters) in a web application, and a passive attacker who inspects network traffic and wants to obtain information about the user's actions.

When a user performs an action on the web application, this causes messages (which we call *web-objects*) to be exchanged between the client- and the server-side of the web application, and the attacker observes bursts of network packets corresponding to each web-object. Traffic is protected by encrypting the data at a certain layer of the protocol stack. Commonly used protection mechanisms are HTTPS, an encrypted connection to proxies (e.g. an SSH tunnel), and encryption of wireless traffic (e.g. using WPA2). Depending on the used protection mechanism, attackers will have a different view of the traffic, as illustrated in Figure 1.

2.2 Basic model

We model the network traffic corresponding to a web application using a directed labeled graph. In this graph, vertices correspond to the states of the web application, edges correspond to possible user actions, and vertex labels correspond to induced traffic patterns.

Definition 1. A *web application* is a directed graph $G = (V, E)$, together with an *application fingerprint* $f_{app} : V \to O$ that maps vertices to *observations* $O \subseteq (A \times \{\uparrow, \downarrow\})^*$, where A is the set of observable objects. We denote the set of paths in G by $Paths(G)$.

An application fingerprint of a vertex is intended to capture an eavesdropper's view of transmitted packets, requests and responses, which necessarily includes the packets' directions. If traffic is not encrypted, we set the observable objects to be bit-strings, i.e. $A = \{0, 1\}^*$. If (a part of) an object is encrypted, only its size remains visible, i.e., we assume that encryption is perfect and length-preserving. Formally, encryption is defined as a function $enc : \{0, 1\}^* \to \mathbb{N}$, and if all objects are encrypted, then we set $A = \mathbb{N}$.

For example, a sequence $(10, \uparrow), (10, \uparrow), (20, \downarrow), (32, \downarrow)$ captures an exchange of encrypted objects, two of size 10 sent from the client to the server, and two of size 20 and 32, respectively, sent from the server to the client.

We next show how Definition 1 can be instantiated to capture two representative scenarios, which we use as running examples throughout the paper. The first scenario models web navigation, where user actions correspond to following hyperlinks. The second scenario models an autocomplete form, where user actions correspond to typing characters into an input field. For ease of presentation, in the following examples we cast the states of a web application as the requested web-objects, and assume that all objects are encrypted.

Example 1. Consider a user who is navigating the Web by following hyperlinks. The web application is a *web-graph* $G = (V, E)$, where each vertex $v = (w_1, w_2, w_3, \dots)$ is a sequence of web-objects, i.e $V \subseteq W^*$ for a set W of web-objects. For example, we may have a vertex v = (a.html,

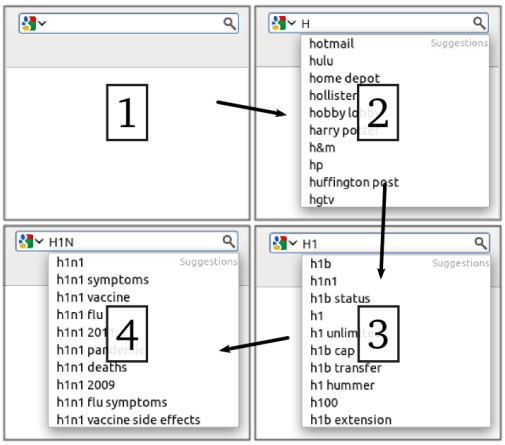


Figure 2. An auto-complete input field

style.css, script.js, image.jpg, video.flv), which corresponds to a particular webpage. An edge (u, v) models that the webpage v can be reached from webpage u by following a hyperlink. If we additionally allow users to jump to arbitrary webpages, the resulting graph will be complete. For the definition of the application fingerprint f_{app}, let the size of a web-request corresponding to a web-object, and the size of the web-object be given as functions $r\colon W \to \mathbb{N}$ and $s\colon W \to \mathbb{N}$, respectively. Then, the application fingerprint of a webpage $v = (w_1, \ldots, w_n)$ is given by

$$f_{app}(v) = ((r(w_1), \uparrow), (s(w_1), \downarrow), \ldots, (r(w_n), \uparrow), (s(w_n), \downarrow)) \,.$$

Example 2. Consider an auto-complete input field where, after each typed character, a list of suggestions is generated by the server and displayed to the user (see Figure 2). Let Σ be the input alphabet (i.e. the set of possible user actions) and Σ^* the set of possible words. We define the corresponding web application as a graph $G = (V, E)$ with $V = \Sigma^*$, the root $r = \varepsilon$ is the empty word, and $(u, v) \in E$ if and only if $v = u\sigma$, for some $\sigma \in \Sigma$. Note that G is in fact a prefix tree [9], and the leafs of the tree form the input dictionary $D \subseteq \Sigma^*$. Let the auto-complete functionality be implemented by a function $suggest\colon V \to S^*$ returning a list of suggestions from the dictionary of possible suggestions $S \subseteq \Sigma^*$. Let the sizes of a suggestion request and the corresponding suggestion list be given as functions $r\colon V \to \mathbb{N}$ and $s\colon S^* \to \mathbb{N}$, respectively. Then, given a word $v \in V$, we define its application fingerprint as

$$f_{app}(v) = ((r(v), \uparrow), (s(suggest(v)), \downarrow)).$$

To capture modifications of observations when traffic passes through different network protocols, as well as the effect of (potentially randomized) countermeasures, we introduce the notion of a network fingerprint.

Definition 2. A *network fingerprint* is a function

$$f_{net}\colon O \to (O \to [0, 1]),$$

such that for each $o \in O$, $\sum_{o' \in O} f_{net}(o)(o') = 1$.

A network fingerprint $f_{net}(o)(o')$ models the conditional probability of outputting a burst of network packets o', given as input a burst of network packets o. We cast network fingerprints using function notation because this allows for a clean combination with application fingerprints.

We now show how the combination of the web application with the network fingerprint f_{net} can be cast as an information-theoretic channel mapping execution paths $Paths(G)$ to sequences of network observations O^*.

Definition 3. Let G be a web application with fingerprints f_{app} and f_{net}. Let X be a random variable with $range(X) = Paths(G)$, and Y a random variable with $range(Y) = O^*$. Then the *traffic channel* induced by (G, f_{app}, f_{net}) is the conditional distribution

$$P[Y = o_1 \ldots o_\ell | X = v_1 \ldots v_\ell] = \prod_{i=1}^{\ell} f_{net}(f_{app}(v_i))(o_i)$$

With Definition 3, we make two implicit assumptions: First, we assume that when a user traverses a path of length ℓ in a web application, the attacker can see a sequence $o_1, \ldots, o_\ell \in O^\ell$ of sequences of packet sizes and directions. This corresponds to an attacker that can distinguish between bursts of traffic corresponding to different vertices, but that cannot observe the timing of the packets (See Section 5 for a treatment of timing). Second, by multiplying the probabilities of observations, we assume that the network bursts corresponding to individual vertices are pairwise independent. Note that this assumption can be weakened by conditioning on multiple vertices, which we forgo for simplicity of presentation. Finally, note that Definition 3 does not make any assumptions on the distribution of X, which models the user's behavior. We will only make such assumptions in Section 6, where we need them for the algorithmic derivation of security guarantees.

3 Information leaks in web-traffic

In this section, we present techniques for quantifying the information that a web application's encrypted traffic reveals about the user's behavior. We base our development on the notion of an information-theoretic channel, which enables us to rely on established notions of confidentiality with their well-understood theory.

3.1 Quantifying information leaks

We model the (secret) behavior of a user as a random variable X and the observable traffic of a web-application

as a random variable Y. The dependency between X and Y is given in terms of a traffic channel $P[Y|X]$ (formalized in Definition 3); Section 6 discusses possible instantiations for $P[X]$.

We characterize the security of this system in terms of the difficulty of guessing the value of X when only the value of Y is known. This difficulty can be captured in terms of information-theoretic entropy, where different notions of entropy correspond to different kinds of guessing. We begin by introducing Shannon entropy and discuss alternative notions below.

Definition 4. The *(Shannon) entropy* of X is defined as

$$H(X) = -\sum_x P[X = x] \log_2 P[X = x],$$

and captures the *initial uncertainty* about the value of X. The *conditional entropy* $H(X|Y)$ of X given Y is defined by

$$H(X|Y) = \sum_y P[Y = y] H(X|Y = y),$$

and captures the *remaining uncertainty* about the value of X when the outcome of Y is known. The *leakage* from X to Y is the reduction in uncertainty about X when Y is observed, i.e. $H(X) - H(X|Y)$.

Shannon entropy is interesting as a measure of confidentiality because one can use it to give a lower bound for the expected number of steps that are required for determining the value of X by brute-force search. Observe that the optimal strategy for this is to try all values of X in order of their decreasing probabilities. For this, let X be indexed accordingly: $P[X = x_1] \geq P[X = x_2] \geq \ldots$. Then the expected number of guesses (also called *guessing entropy* [34]) required to determine X is defined by $G(X) = \sum_{1 \leq i \leq |X|} i\, P[X = x_i]$, and the conditional version $G(X|Y)$ is defined in the natural way [28]. The following result [28, 34] bounds $G(X|Y)$ in terms of Shannon entropy.

Proposition 1. $G(X|Y) \geq \frac{1}{4} 2^{H(X|Y)}$

Note however that for heavily skewed X, the *expected* number of guesses required to determine X can be large, even if an attacker has a considerable chance of correctly guessing the value of X in one attempt. To see this, consider a random variable X distributed by $P[X = x_1] = \frac{1}{2}$ and $P[X = x_i] = \frac{1}{2n}$, for $i \in \{2, \ldots, n\}$: the expected number of guesses to determine X grows linearly with n, but the probability of correct guessing in one shot remains $\frac{1}{2}$.

The *min-entropy* H_∞ (see, e.g., [39] for a definition) accounts for such scenarios by capturing the probability of correctly guessing the secret in *one* attempt, i.e., it delivers worst-case rather than average-case guarantees. The *g-entropy* [1] H_g generalizes min-entropy by parameterizing the notion of an attacker's gain.

The model presented in Section 4 can be used in conjunction with all $\mathcal{H} \in \{H, H_\infty, H_g\}$, and we will keep our presentation parametric. The algorithmic results in Section 6 rely on the so-called chain rule, which is only satisfied by Shannon entropy. I.e., there we trade the strength of the security guarantees for the tractability of the analysis.

4 Network fingerprints

In this section, we take a closer look at network fingerprints. In particular, we show that their composition decreases the side-channel leakage of a web application. This result has two implications: First, it enables us to compose countermeasures against side-channel attacks, obtaining assertions about their relative strengths for free. We utilize this result by giving a toolkit of basic countermeasures, and we show how they can be combined to encompass proposals for countermeasures that appear in the recent literature. Second, it enables us to establish the soundness of our bounds with respect to observers of lower layers of the protocol stack.

4.1 Network fingerprint composition

We now show how network fingerprints can be composed, and how this composition affects the security of a web application.

Definition 5. Given network fingerprints $f_{net} \colon O \to (O' \to [0, 1])$ and $f'_{net} \colon O' \to (O'' \to [0, 1])$, we define the *composed network fingerprint* $(f'_{net} \circ f_{net}) \colon O \to (O'' \to [0, 1])$ by

$$(f'_{net} \circ f_{net})(o)(o'') = \sum_{o' \in O'} f_{net}(o)(o') f'_{net}(o')(o'') \,.$$

Note that composition of network fingerprint corresponds to concatenating two conditional distributions and marginalizing the intermediate results. In the literature, the resulting conditional distribution has been referred to as a *channel cascade* [20].

The following result states that fingerprint composition does not decrease a web application's security. For the formal statement, let f_{net} and f'_{net} as in Definition 5, let X be a random variable with $range(X) = Paths(G)$, Y a random variable with $range(Y') = (O')^*$ induced by f_{net}, and Y'' a random variable with $range(Y'') = (O'')^*$ induced by the composition $f'_{net} \circ f_{net}$.

Theorem 1. *$\mathcal{H}(X|Y'') \geq \mathcal{H}(X|Y')$, for $\mathcal{H} \in \{H, H_\infty, H_g\}$.*

For the proof of Theorem 1, we use the data processing inequality. Intuitively, this inequality states that computation on data does not increase the amount of information contained in this data. This inequality is a well-known

result for Shannon-entropy [11], and also holds for min-entropy [20] and g-leakage [1].

Lemma 1 (Data Processing Inequality)**.** *Let X, Y, Z be random variables such that $P[Z|XY] = P[Z|Y]$, i.e. X, Y, Z form a Markov chain. Let $\mathcal{H} \in \{H, H_\infty, H_g\}$. Then*

$$\mathcal{H}(X|Z) \geq \mathcal{H}(X|Y) .$$

Proof of Theorem 1. By definition of fingerprint composition, the output of Y'' only depends on the output of Y', i.e X, Y', and Y'' form a Markov chain. This allows application of Lemma 1, which concludes the proof.

□

From Theorem 1 it follows that given a fingerprint f_{net}, composing other fingerprints to the left always yields better security. We conclude this section with the observation that this is not always the case for composition to the right, i.e. there are cases when $f_{net} \circ f'_{net}$ induces a lower remaining uncertainty than f_{net} (see also [20]).

4.2 Basic network fingerprints

In this section, we define a number of basic network fingerprints that will form the basis for the construction of countermeasures by composition, and for the modeling of the attacker observations at different layers of the protocol stack. In Appendix A, we discuss practical aspects of the implementation of those network fingerprints as countermeasures.

By Definition 2, a network fingerprint is applied to an observation corresponding to the currently visited vertex, which is a sequence of observable objects and directions in $O = (A \times \{\uparrow, \downarrow\})^*$. For convenience of notation, in this section we denote sequences of observable objects as $\vec{o}$, and define some basic fingerprints on single observable objects $o = (m, d)$ from $A \times \{\uparrow, \downarrow\}$, where m is the observed message and d is its direction. Single observable objects are extended to sequences of observable objects by applying them independently to each object in the sequence. Formally, this corresponds to taking the product of the corresponding distributions. Furthermore, we denote the probability $f_{net}(o)(o')$ in random variable notation: $P[F' = o'|F = o]$. In the following, we elaborate on each of the basic network fingerprints.

4.2.1 Padding

Padding is the most commonly considered countermeasure in the literature (see, e.g., [7, 30, 40]). It changes observations $o = (m, d)$ by (randomly) increasing the size of the message $|m|$ and retaining its direction d; formally, it can be defined as a conditional distribution $P[F'_{pad}|F_{pad}]$ with

$$P[F'_{pad} = (m', d')|F_{pad} = (m, d)] > 0 \implies d' = d \wedge |m'| \geq |m| .$$

4.2.2 Dummy

The network fingerprint *dummy* adds redundant files to observations. Formally, we define *dummy* as a conditional distribution with

$$P[F'_{dum} = \vec{o}'|F_{dum} = \vec{o}] > 0 \implies \vec{o} \sqsubseteq \vec{o}' ,$$

i.e. the inputs $\vec{o}$ form a subsequence of the outputs $\vec{o}'$.

4.2.3 Split

The network fingerprint *split* causes splitting a file into smaller files. Formally, we define split as a conditional distribution with

$$P[F'_{split} = (m_1, d_1), \ldots, (m_n, d_n)|F_{split} = (s, d)] > 0 \implies \forall i. d_i = d \wedge \sum_{i=1}^{n} |m_i| = |m| .$$

4.2.4 Shuffle

In a webpage containing more than one objects, the base .html file is always requested first, and the remaining web-objects are usually downloaded in the order in which they are included into the .html file. Ordered sequences of packet sizes can be used for identifying websites, e.g. see [32]. In the following, we propose the *shuffle* network fingerprint, where objects will be requested in an arbitrary order. Formally, *shuffle* is defined as a conditional distribution with

$$P[F'_{shuffle} = \vec{o}'|F_{shuffle} = \vec{o}] > 0 \implies \vec{o}' \in \Pi(\vec{o}) ,$$

where $\Pi(\vec{o})$ is the set of all permutations of the sequence $\vec{o}$. For an example implementation, see Appendix A.

4.2.5 k-parallelism

We define the k-parallelism (or *k-par*) network fingerprint that captures the addition of redundant sequences of observations, corresponding to $k-1$ vertices from G; thus, an observer does not know which of the observations correspond to the actual vertices, and which to the redundant ones. We formally define *k-par* as a conditional distribution with

$$P[F'_{k-par} = (o_1, \ldots, o_k)|F_{k-par} = o] > 0 \implies \exists i \in [k]. o_i = o .$$

4.2.6 Interleave

When inspecting k concurrent streams of observations, e.g. corresponding to traffic of k users or the result of the k-par countermeasure, an observer may be able to directly infer which sequence of observations corresponds to which sequence of downloaded files (e.g. by inspecting TCP headers of TLS-encrypted data). The *interleave* network fingerprint

(or *inter*) turns k consecutive sequences of observations into one sequence in which the observations of k sequences appear interleaved, but without changing the relative order of observations within each sequence. We forgo a formal definition of *inter*.

4.3 Building composed countermeasures

The basic network fingerprints defined above can be used as building blocks for complex countermeasures, which can be obtained using network fingerprint composition (see Definition 5). We illustrate this using two examples from the recent literature: The first one corresponds to one of the modules used in the HTTPOS system [33], the second one corresponds to the traffic morphing countermeasure [41].

4.3.1 HTTPOS

As part of HTTPOS, Luo et al. [33] propose a method which (1) injects requests for objects, and (2) uses the TCP maximum segmentation size (MSS) and advertising window options to reduce packet sizes.

Part (2) works in two stages: First, the first m bytes of an object are split into equal-sized packets. Second, the remaining n bytes of the object are split into two packets of size r and $n - r$, respectively, for a random $r < n$. Part (2) can be cast in terms of the *split* network fingerprint, and part (1) can be formally captured by the *dummy* network fingerprint. Combining both countermeasures by fingerprint composition, we obtain

$$dummy \circ split$$

as a formalization of the proposed HTTPOS module. A simple consequence of applying Theorem 1 to this formalization is that that the module indeed offers better security than using no protection, or *split* alone.

4.3.2 Traffic morphing

Wright et al. [41] propose traffic morphing as a countermeasure against side-channel attacks. The idea behind traffic morphing is to modify a website's profile (i.e., distribution of packet sizes and numbers) by splitting or padding packets with the goal of mimicking another website's profile. This countermeasure can be cast in our model by a suitable combination of the *split* and *pad* countermeasures.

4.4 Soundness with respect to the protocol stack

By choosing a particular application fingerprint, we model an adversary who can monitor traffic at a specific protocol layer (e.g. TCP packets), and we assume that the payload of the corresponding messages is encrypted at some higher layer. From the perspective of physical security and verification, it is not immediately clear whether the security guarantees based on this model extend to adversaries who can monitor lower layers of the protocol stack, e.g., the physical layer: For example, proofs of semantic security of a cryptosystem do not imply security against an adversary who can monitor timing and power consumption [26]. Likewise, in program verification, formal proofs of correctness and security at the level of source code do not immediately imply correctness or security of the corresponding machine code.

In this section, we show that the situation is different for the traffic analysis problem we consider. There, the security guarantees obtained in our model are preserved when moving to a lower layer of the protocol stack. We obtain this result by casting the standard operations required for this (i.e. segmentation, adding packet headers, and tunneling) as fingerprints according to Definition 2. Theorem 1 then implies that our security guarantees translate to lower layers of the stack.

Packet headers When passing through different network layers, each layer adds its own headers. Such headers can be interpreted as a padding

$$header = pad\ .$$

If the header is of fixed size, then *pad* will be deterministic.

Packet segmentation Packet segmentation is the process of dividing a packet into smaller chunks, and each of those chunks obtains a new header. For example, segmentation is used by the TCP protocol for congestion control. Formally, packet segmentation corresponds to

$$segment = header \circ split\ ,$$

where *split* is usually deterministic.

Tunneling Tunneling encapsulates traffic from one protocol into another protocol. It may add a new layer of encryption (e.g. in the case of a SSH tunnel), in which case none of the original packet headers will be transmitted in the clear (see Section 2.1). As a consequence, an attacker may lose the ability to distinguish which packets correspond to which web object; additionally, background traffic, which is easily filtered out in the presence of unencrypted headers, may now appear to be part of the downloaded content. Thus we model encrypted tunneling as

$$tunnel = inter \circ segment \circ k\text{-}par\ ,$$

which corresponds to an interleaving of concurrent streams of segmented packets.

5 Advanced features

The model presented so far does not encompass caching or timing behavior and, consequently, does not offer protection against side-channels induced by these features. In this section we discuss how our model can be extended with these advanced features. Moreover, we show how our model allows for specifying that only a part of the user's input is sensitive.

5.1 Timing behavior

For closing information leaks due to packet delays, one can make use of the techniques for predictive mitigation of timing channels proposed in [2]. The mitigation scheme buffers packets and permits them to enter the network only at multiples of a particular time quantum q. With this restriction, the only source of timing leakage are the points in time at which *no* packet is sent. The core idea of predictive mitigation is to double the time delay between packets after such time points, which ensures that their number (and hence the leakage) remains bounded within each time interval. The uncertainty bounds obtained by timing mitigation are of the form

$$\mathcal{H}(X|Z) \geq \mathcal{H}(X) - O(\log_2(T+1)) \,,$$

where X is secret user input, Z is an abstraction of the observable timing behavior, T is the number of observed time units, and $\mathcal{H} \in \{H, H_\infty, H_g\}$. For *Shannon* entropy, the guarantees obtained for the timing channel compose with the guarantees obtained for the traffic channel, as shown in [2]:

$$H(X|YZ) \geq H(X|Y) - O(\log_2(T+1)) \,,$$

where $H(X|YZ)$ denotes the remaining uncertainty from user behavior X to an adversary who can simultaneously observe packet sequences Y and their timing Z. This result enables us to combine our bounds for traffic channels with bounds for timing channels and obtain a guarantee that accounts for both. The proof of this result relies on the chain rule and does not immediately extend to H_∞ and H_g.

5.2 Modeling browser caches

A browser cache stores downloaded web-objects so that if an object is requested a second time, it can be fetched from the cache, avoiding costly communication with the server. We now show how the definitions presented in Section 2.2 can be extended to take into account the effect of caching. As in Example 1, we assume vertices are sequences of web-objects, i.e. $V \subseteq W^*$.

For the modeling of the traffic induced by a webpage on a given execution path, a browser cache has the following effects: (1) the sequence of downloaded web-objects is a subsequence of the web-objects contained in the webpage, and (2) all web-objects contained in the page have been downloaded at some point in the execution path. Both effects are captured by the following definition.

Definition 6. A *cached request* is a function $req\colon V^+ \to W^*$ that takes execution paths $v_1, \ldots, v_\ell \in Paths(G)$ and returns a subset of the web-objects of the terminal vertex v_ℓ.

1. $req(v_1, \ldots, v_\ell) \sqsubseteq v_\ell$, and
2. $set(v_\ell) \subseteq \bigcup_{i=1}^{\ell} set(req(v_1, \ldots, v_i))$,

where $\sqsubseteq$ denotes the subsequence relation, and $set(\cdot)$ denotes the set representation of a sequence.

Notice that conditions 1 and 2 define a family of cache behaviors (rather than a unique one) for which we implicitly assume that the cache is empty when the browsing starts. We can now extend Definition 3 to capture traffic channels with cache.

Definition 7. Let G be a web application with fingerprints f_{app} and f_{net} and a browser cache modeled by req. Let X be a random variable $range(X) = Paths(G)$, and Y a random variable with $range(Y) = O^*$. Then the *traffic channel* induced by $(G, f_{app}, f_{net}, req)$ is the conditional distribution

$$P[Y = o_1 \ldots o_\ell | X = v_1 \ldots v_\ell] = \prod_{i=1}^{\ell} f_{net}(f_{app}(req(v_1, \ldots, v_i))(o_i)$$

Informally, Definition 7 states that the traffic induced by each vertex depends on the entire input path, but that the outputs of the individual network are mutually independent, given the web-objects requested by the browser.

5.3 Specifying secrets

The measures of security introduced in Section 3.1 capture bounds on the difficulty of guessing the exact value of X, which corresponds to the entire path taken by the user. In many cases, however, one is not interested in protecting the entire path, but rather some sensitive property of the path, e.g. containment in a critical subset of vertices.

To capture such properties in our model, we specify the secret as a function $sec\colon Paths(G) \to range(sec)$ of arbitrary range that takes as input an execution path and returns the sensitive information contained in it. From the perspective of information-flow security, sec can be thought of as a specification of the *high* parts of a path. The model from Section 2 can be straightforwardly extended to account for a specification of secrets sec by replacing X with the random variable $X_{sec} = sec(X)$.

Examples of relevant instances of sec are the identity function $sec = id_{Paths(G)}$ (which models that the whole path is sensitive), a function $sec(v_1, \ldots, v_\ell) = \{v_1, \ldots, v_\ell\}$ returning a set-representation of a path (which models that the set

of visited vertices is sensitive, but not their ordering), or arbitrary binary predicates, such as $sec(v_1 \dots, v_\ell) = true$ iff $v \in \{v_1, \dots, v_\ell\}$ (which models that it is sensitive whether or not a vertex v has been visited). Secret specifications can be also used to model the scenario where a user wants to hide the currently visited *website*, rather than pages within this website, e.g. if traffic is forced through an SSH tunnel (see also Section 2.1). Then, we define $G = (V, E)$ as the Internet-graph, and $V_1, \dots, V_k$ as the partition of V into disjoint blocks corresponding to k different websites. The sensitive information in this scenario is captured by the function *sec*, which returns the ℓ blocks corresponding to the visited websites.

6 Algorithms for practical evaluation of web applications

So far we have introduced techniques for reasoning about the *relative* strength of countermeasures (Theorem 1). In this section we devise techniques for the derivation of *absolute* security guarantees, i.e. concrete numbers for the remaining uncertainty $\mathcal{H}(X|Y)$ about the user input. For this, two challenges need to be addressed.

The first is that the derivation of absolute guarantees requires making assumptions about (the attacker's initial uncertainty $\mathcal{H}(X)$ about) the user's behavior. The second is that the direct computation of $\mathcal{H}(X|Y)$ requires enumerating the values of X (i.e. all paths) and Y (i.e. all possible traffic patterns), which quickly becomes infeasible. In this section we present algorithms that enable the efficient computation of formal security guarantees. Our algorithms assume Markov models of user behavior and rely on the chain rule of entropy, which is only satisfied by Shannon entropy $H(X)$. As a consequence, their application is restricted to scenarios where such assumptions can be justified and where guarantees based on Shannon entropy (see Section 3) are adequate.

We begin by showing how, under these assumptions, the initial uncertainty about a user's navigation in a website can be computed. We then present a generic approach for constructing countermeasures based on bisimulations and for computing the uncertainty induced by such countermeasures.

6.1 Modeling user behavior as a Markov chain

We capture user behavior by a random variable X, where $P[X = (v_1, \dots, v_\ell)]$ describes the probability of the user taking execution path $(v_1, \dots, v_\ell)$, see Section 2. The random variable X can be decomposed into the components $X_1, \dots, X_\ell$ corresponding to the vertices on the path. When the user's choice of the next vertex depends only on the currently visited vertex, then $X_1, \dots, X_\ell$ form a *Markov chain*. Formally: $P[X_{i+1} = v_{i+1} | X_i = v_i, \dots, X_1 = v_1] = P[X_{i+1} = v_{i+1} | X_i = v_i]$ for $i \in \{1, \dots, \ell - 1\}$. Then, we obtain

$$P[X = (v_1, \dots, v_\ell)] = P[X_1 = v_1] \prod_{i=1}^{\ell-1} P[X_{i+1} = v_{i+1} | X_i = v_i] \,.$$

This decomposition enables one to specify the probability of execution paths in terms of probabilities of individual transitions. The Markov property is clearly valid in the auto-complete input fields from Example 2 where the execution paths form a tree, and it also is a commonly made assumption for predicting navigation behavior of users [19, 38].

6.2 Computing the initial uncertainty

We next devise an algorithm for computing the initial uncertainty $H(X)$ about the behavior of a user, based on the stationary distribution of X and the chain rule of entropy. We then show how X can be instantiated and how its stationary distribution can be computed using the PageRank algorithm.

6.2.1 Initial uncertainty based on stationary distributions

From the Fundamental theorem of Markov chains [35] it follows that each finite, irreducible, and aperiodic Markov chain converges to a unique stationary distribution. Formally, if the Markov chain is given in terms of a transition matrix Q, where $q_{i,j}$ denotes the probability of moving from v_i to v_j, there is a row vector (the *stationary distribution*) π with $\pi = \pi Q$.

Under the above conditions, the user's choice of a next vertex will converge to π, which we use to capture the probability of the user choosing the *first* vertex $P[X_1 = v_1]$. The following theorem gives a handle on efficiently computing the initial uncertainty about the user's execution path.

Theorem 2. *Let $X_1, \dots, X_\ell$ be a Markov chain with $P[X_1] = \pi$, where π is the stationary distribution. Then,*

$$H(X_1, \dots, X_\ell) = H(X_1) + (\ell - 1) H(X_2 | X_1).$$

Proof.

$$\begin{aligned} & H(X_1, \dots, X_\ell) \\ \overset{(*)}{=} \; & H(X_\ell | X_{\ell-1}, \dots, X_1) + \dots + H(X_2 | X_1) + H(X_1) \\ \overset{(**)}{=} \; & H(X_\ell | X_{\ell-1}) + H(X_{\ell-1} | X_{\ell-2}) + \dots + H(X_2 | X_1) + H(X_1) \end{aligned}$$

Here $(*)$ follows from the chain rule for Shannon entropy and $(**)$ follows from the Markov property. As $P[X_1]$ is the stationary distribution, we have $H(X_k | X_{k-1}) = H(X_j | X_{j-1})$ for all $j, k \in \{2, \dots, n\}$, which concludes the proof. □

Given the stationary distribution, Theorem 2 enables the computation of the initial uncertainty $H(X)$ about the user's navigation in time $O(|V|^2)$, for any fixed ℓ. We next show how the Markov chain representation of X can be instantiated, and how its stationary distribution can be obtained, using the PageRank algorithm.

6.2.2 Using PageRank for practical computation of the initial uncertainty

PageRank [36] is a link analysis algorithm that has been applied for predicting user behavior in web usage mining [19]. It relies on the *random surfer model*, which captures a user who either follows a link on the currently visited webpage, or jumps to a random webpage.

Formally, the probability of following links is given by a transition matrix Q', and the probability of random jumping is $1-\alpha$, where the *damping factor* $\alpha \in [0, 1]$ is usually set to 0.85. The user's behavior is then captured by the transition matrix $Q = \alpha Q' + (1 - \alpha)p\mathbf{1}$, where p is a row vector representing the probability of jumping to pages, and $\mathbf{1}$ is the column vector with all 1 entries. One typically assumes uniform distributions for the rows q'_i of Q' and for p, however the probabilities can also be biased according to additional knowledge, e.g. obtained by mining usage logs [18].

Given a transition matrix Q of a website, the PageRank algorithm computes the corresponding stationary distribution π, and π_i is called the PageRank of a webpage i. Notice that the conditions for the existence of π are usually fulfilled: irreducibility is guaranteed by a damping factor $\alpha < 1$, and in practice websites are aperiodic.

6.3 Constructing path-aware countermeasures

Most traffic analysis countermeasures aim to provide protection by making multiple states of a web-application indistinguishable for an attacker who can only observe their traffic fingerprints (e.g. see [41]). In our model, we say that two vertices v_1 and v_2 are *(f)-indistinguishable* for some countermeasure f whenever $f(v_1) = f(v_2)$, i.e. the observations produced by v_1 and v_2 have the same probability distribution. Clearly, f-indistinguishability induces a partition $P = \{B_1, \ldots, B_m\}$ on the state space V of a web-application, where the blocks B_i correspond to the equivalence classes of f-indistinguishability.

Unfortunately, indistinguishability of individual states does not protect information gained by observing a sequence of states. Consider, e.g., a scenario where an attacker observes traffic produced by typing a word in English. Assume that the first typed character is known to be t and the second character is either h or s, which we assume to be f-indistinguishable. From the first action t, the attacker can deduce that the second user action is most likely h, i.e. secret information is revealed.

6.3.1 Ensuring indistinguishability of paths

To protect the information contained in the transition between states, we devise *path-aware countermeasures*, which require that the induced partition be in fact a *bisimulation* [29] (also called *lumping* [25]), a property which ensures behavioral equivalence of states.

Definition 8. A partition $P = \{B_1, \ldots, B_m\}$ of the state space V of a Markov chain $X = (X_1, \ldots, X_\ell)$ is a *(probabilistic) bisimulation* if for any blocks $B_i, B_j \in P$ and for any $v_i, v_h \in B_i$, $\sum_{v_j \in B_j} q_{i,j} = \sum_{v_j \in B_j} q_{h,j}$. We define the corresponding *quotient process* $Y = (Y_1, \ldots, Y_\ell)$ as the random variable with state space P, such that $Y_k = B_i$ iff $X_k = v$ and $v \in B_i$.

A fundamental characteristic of bisimulations is that they preserve the Markov property of the resulting quotient process.

Theorem 3 (Kemeny-Snell [25])**.** *A partition of the state space of a Markov chain is a probabilistic bisimulation iff the corresponding quotient process is a Markov chain.*

To measure the strength of a path-aware countermeasure, we need to calculate the remaining uncertainty $H(X|Y)$, for which we have

$$H(X|Y) \geq H(X) - H(Y), \tag{1}$$

with equality whenever Y is determined by X (e.g., if the corresponding countermeasure is deterministic). By Theorem 3, the resulting quotient process Y is a Markov chain. This enables us to use Theorem 2 for computing $H(Y)$, leading to an efficient algorithm returning a lower bound for $H(X|Y)$.

6.3.2 Implementing path-aware countermeasures

For a given countermeasure f, we seek to group together vertices in V to blocks of indistinguishable vertices, such that the resulting partition is path-aware, i.e. a bisimulation. This could be trivially achieved by padding all possible observations to a maximal element o^*. While the corresponding (1-block) partition achieves maximal security, it may also induce an unacceptably large traffic overhead. Our goal is hence to find a bisimulation that coarsens the partition induced by f and that offers a more attractive trade-off between security and performance.

While there are efficient algorithms for computing bisimulations that *refine* a given partition to a bisimulation, we are not aware of existing algorithms for obtaining bisimulations by coarsening. We tackle the problem by the following two-step approach.

In a first step, we compute a set of *random* bisimulations on V. To achieve this, we select random initial partitions

of V with only two blocks, e.g. by flipping a coin for each vertex and selecting a block accordingly. For each of those two-block partitions, we compute the coarsest bisimulation that refines it using the efficient algorithm by Derisavi et al. [13], which we call `CoarsestBisimulation` in the remainder of this paper.

In a second step, we coarsen the partition given by f to the partition $P = \{B_1, \ldots, B_m\}$ given by each bisimulation computed in the previous step. We achieve this by modifying f to a new countermeasure that is a constant function on each B_i. We consider two such modifications:

1. The first countermeasure, which we call f_{limit}, returns for each vertex $v \in B_i$ the maximum over all vertices $w \in B_i$ of the sum of the sizes of the observable objects for each countermeasure $f_{\text{limit}}(v) = \max_{w \in B_i} \sum_i |o_{i,w}|$, where $f(w) = (o_{1,w}, \ldots, o_{t,w})$. If observations consist of one object each, f_{limit} can be implemented by padding. If observations consist of multiple objects, an implementation may not be possible without additional overhead, and thus f_{limit} is a theoretical concept representing the smallest possible size that can be achieved without losing information about f, and hence represents a lower limit for the cost required by any countermeasure inducing a partition $\{B_1, \ldots, B_m\}$.

2. The second countermeasure, which we call f_{order}, first orders for each vertex $v \in B_i$ the components in $f(v)$ according to their (descending) size, and then pads the k-th component to the maximum size of the k-th components over all $w \in B_i$. This countermeasure can be built from the basic countermeasures *pad*, *dummy*, and *shuffle* (see Section 4.2).

In our experiments in Section 7, we demonstrate the overhead induced by the practical f_{order}, as well as by the cost-limiting f_{limit} countermeasures. The results show that there is room for designing more cost-effective countermeasures which induce an overhead closer to f_{limit}; for this to be achieved, further basic countermeasures can be utilized, e.g. splitting of objects.

7 Case studies

In this section we report on two case studies in which we apply the approach presented in this paper to evaluate the security of countermeasures for a website and an autocomplete field. Moreover, we analyze the overhead in traffic induced by instantiations of the countermeasure to different strengths, which gives a formal account of the ubiquitous trade-off between security and performance in side-channel analysis.

7.1 Web navigation

We analyze a web navigation scenario as described Example 1. As target we use the Bavarian-language Wikipedia[1] of more than 5,000 articles. We instantiate user models and network fingerprints as follows.

User model We assume the random surfer model (see Section 6.2), where a user is navigating through Wikipedia articles. To instantiate the web-graph G, we crawled the website only following links leading to the Wikipedia article namespace, where the crawling was performed using a customized `wget`[2]. Excluding redirect pages, we obtained a graph with 3,496 vertices.

Application fingerprint For each vertex v, we took the application fingerprint $f_{app}(v)$ to be the tuple containing the size of the source .html file and the sizes of the contained image files. On average, the total size of each vertex was 104 kilobytes, with a standard deviation of 85 kilobytes. Taking sizes of web-objects instead of sizes of packets is a safe approximation whenever encryption happens at the object-level, see Section 4.4. We did not account for the sizes of requests because they are almost of equal size in our example, and padding them to a uniform constant size is cheap.

Results without countermeasure We compute the initial uncertainty $H(X)$ about the user's navigation using Theorem 2, where we obtain the stationary distribution of the user's starting point using PageRank, as described in Section 6.2. The first row in Figure 3 gives the values of $H(X)$ for varying path lengths ℓ. An interpretation of the data in the sense of Proposition 1 shows that for $\ell = 5$ we already obtain a lower bound of around 2^{30} for the expected number of guesses to determine the correct execution path. For the corresponding sequence of network fingerprints Y (without any countermeasure applied) and $\ell > 1$, we consistently obtain a remaining uncertainty $H(X|Y)$ of 0. This follows because in our example almost every page is determined by its fingerprint, and whenever there is ambiguity, the page can be recovered from observing the path on which it occurs. Also note that a naive computation of $H(X)$ quickly becomes infeasible, because the number of paths is $|V|^\ell$, due to the completeness of the graph induced by the random surfer model.

Results with countermeasure As described in Section 6.3, we obtain a number of path-aware countermeasures by refining randomly chosen initial partitions. For

[1] `http://bar.wikipedia.org`
[2] `http://www.gnu.org/software/wget/`

ℓ	1	3	5	9	15	25	40
$H(X)$	10.1	21	31.8	53.4	85.9	139.9	221
# paths	3496	$2^{36.5}$	$2^{59.8}$	2^{106}	2^{176}	2^{295}	2^{472}

Figure 3. Web navigation: the initial uncertainty $H(X)$ about the user's navigation in the regional-language Wikipedia for a varying path length ℓ. # paths denotes the number of execution paths.

each of them, we evaluate the effect on security and performance: (1) we compute the remaining uncertainty $H(X|Y)$ using Equation 1 and Theorem 2; (2) we compute the relative expected overhead of a countermeasure f_{net} as $E_v[\#(f_{net}(v)) - \#(f_{app}(v))]/E_v[\#(f_{app}(v))]$, where $\#(\cdot)$ denotes the size of the payload, i.e. the sum of the number of bytes used for the individual observations. The results below are obtained for paths of length $\ell = 5$ and the practical f_{order} and cost-limiting f_{limit} countermeasures (see Section 6.3).

As reference point we use the countermeasure that makes all states indistinguishable and induces maximal security, i.e. $H(X|Y) = H(X)$. On this countermeasure, f_{order} produces a relative expected overhead of 73.5, while f_{limit} produces a relative expected overhead of 9.7. Trading security for performance, we randomly chose 500 initial partitions, and refined them to path-aware countermeasures using `CoarsestBisimulation` (see Section 6.3). The remaining uncertainty and the relative expected overhead for each of those countermeasures are depicted in Figure 4(a) (for f_{order}), and Figure 4(b) (for f_{limit}). The results spread from partitions delivering strong security at the price of a high overhead, to smaller overhead at the price of weaker security.

Figure 5 details on 8 of the 500 computed bisimulation partitions. The first and the last lines present the trivial coarsest and finest partitions, respectively. The data shows that, when performance is mandatory, one can still achieve 6.63 bits of uncertainty (corresponding a lower bound of 25 expected guesses, according to Proposition 1) with an overhead of factor 2.75. On the other hand, stronger security guarantees come at the price of overhead of more than factor 10. The high required overheads in the web-navigation scenario can be explained by the large diversity of the sizes and numbers of the web-objects in the studied pages. As the next case study shows, the required overheads are much lower for web applications with more homogeneous web-objects.

7.2 Auto-complete input field

We analyze the scenario of an auto-complete input field as described in Example 2. Unlike graphs corresponding

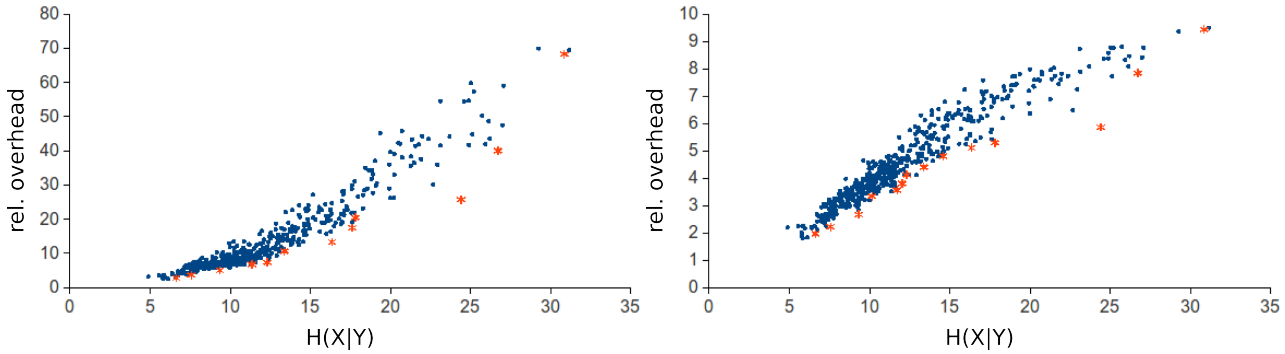
$H(X\|Y)$	overhead f_{order}	overhead f_{limit}
31.77	73.47	9.71
26.7	40.01	7.83
24.4	25.68	5.86
16.35	13.29	5.11
12.32	7.31	4.11
9.35	5.04	3.18
6.63	2.78	1.97
0	0	0

Figure 5. Web navigation: the remaining uncertainty $H(X|Y)$ about the user's navigation and the relative expected overhead for 8 selected bisimulation partitions.

to websites, the graph corresponding to an auto-complete field is a tree with a designated root and a small number of paths. This allows us to compute the remaining uncertainty by enumerating the paths in the tree, which is why we can also consider min-entropy H_∞ (see Section 3). In this case study, we instantiate user models and network fingerprints as follows.

User model As described in Section 6.1, user behavior is characterized by a probability distribution $P[X = (v_1, \ldots, v_\ell)]$ of possible paths of length ℓ, which for auto-complete input fields corresponds to the set of words of length ℓ. To obtain such a distribution, we take as an input dictionary D the 1,183 hyponyms of (i.e., terms in an *is-a* relationship with) the word "illness", contained in the WordNet English lexical database [21]. We use the number of results of a corresponding Google query as an approximation of the probability for each term in the dictionary: We issue such queries for all 1,183 words in the dictionary D and count their relative frequencies, thereby establishing probabilities for the leaves in the prefix tree corresponding to D. We traverse the tree towards the root, instantiating the probability of each *vertex* as the sum of the probabilities of its children. Then we instantiate the probabilities of the outgoing *edges* of each vertex with values proportional to the probabilities of its children.

Application fingerprint We instantiate the application fingerprints as follows. The size of the request $r(v)$ (see Example 2) is given by the length of the typed word v. The size of the response $s(suggest(v))$ is characterized by the size of the suggestion list given by the Google's auto-complete service on query v. We issued queries for all 11,678 vertices in the prefix tree to instantiate these numbers. The responses in all cases consisted of only one packet with average size of 243 bytes and a standard deviation of 97 bytes.



(a) Results with the f_{order} countermeasure

(b) Results with the f_{limit} countermeasure

Figure 4. Web navigation: the remaining uncertainty $H(X|Y)$ about the user's navigation in the regional-language Wikipedia versus the relative expected overhead, using the f_{limit} and f_{order} countermeasures, for 500 randomly chosen initial partitions. Red stars represent favored resulting partitions, i.e. resulting in a better security-versus-overhead trade-off than close-by points.

Results without countermeasure We compute initial uncertainty about the typed word as described in Section 3. For a varying path length $\ell = 1, \ldots, 7$, the initial uncertainty is between 3.79 and 5.65 bits of Shannon Entropy, and between 1.61 and 2.85 of min-entropy (see Figure 6). Unlike graphs corresponding to websites, the graph corresponding to an auto-complete field has a tree-structure, and the number of paths is bounded by the number of terms in the dictionary D. The initial uncertainty is highest for paths of a smaller length, and the decrease of the uncertainty for longer paths occurs because the transition relation reveals more information: there are less words in D sharing the same prefix. In all cases the remaining uncertainty after observing the traffic was 0, meaning that in those cases the web application is highly vulnerable to attacks: an attacker can infer the secret input word with probability 1.

ℓ	1	2	3	4	5	6	7
$H(X)$	4.22	5.09	5.52	5.65	5.52	5.51	5.32
$H_\infty(X)$	2.7	2.85	2.65	2.39	1.96	1.71	1.61
# paths	47	238	538	657	710	773	804

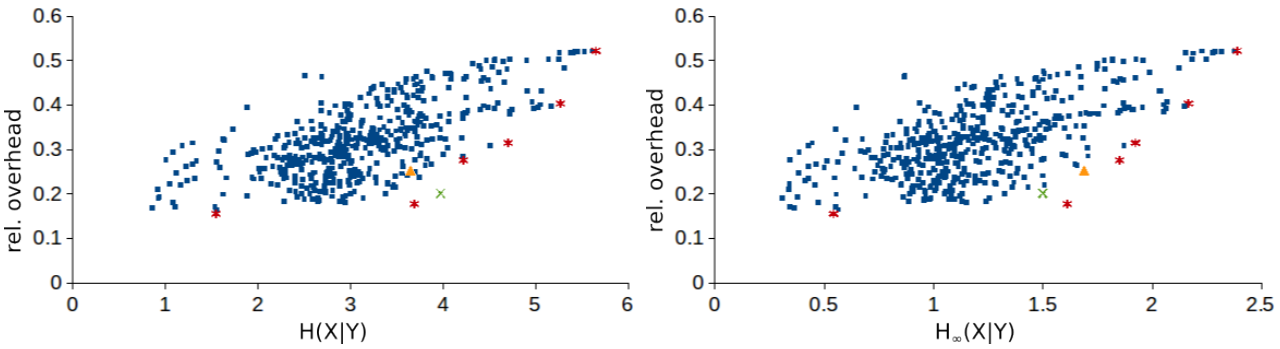
Figure 6. Auto-complete: the initial uncertainty (in terms of Shannon entropy $H(X)$ and min-entropy $H(X|Y)$) about the typed word in the auto-complete input field for a varying path length ℓ. # paths denotes the number of execution paths.

Results with countermeasure We also measured the vulnerability of the auto-complete field with countermeasures applied. States were made indistinguishable by applying f_{limit}, which here can be practically implemented because the observations consist of single files (see Section 6.3). In the following, we report on our experiments for a path length of $\ell = 4$, where we obtain the following results:

First, padding all vertices to a uniform size results in a maximal uncertainty of 5.65 bits of Shannon entropy and 2.39 bits of min-entropy, respectively, with an relative expected overhead of 2.9. Second, padding only vertices at the same depth to a uniform size, the relative expected overhead drops to 0.52. The resulting uncertainty remains maximal because, in our model, the depth of a vertex is common knowledge. Finally, trading security for performance, we construct path-aware countermeasures using `CoarsestBisimulation`, for 500 randomly chosen initial partitions (see Section 6.3). Figure 7 depicts the trade-off between remaining uncertainty and relative expected overhead in this case. For example, the data show that the overhead needed for maximal protection can be decreased by 46%, for the price of leaking of 1.44 bits of Shannon entropy (0.54 bits of min-entropy), and by 65%, for the price of leaking 1.97 bits of Shannon entropy (0.78 bits of min-entropy). Note that in most cases the two considered entropy measures favor the same partitions, as depicted by the partitions corresponding to the red stars in Figure 7(a) and Figure 7(b) coincide. This is not always the case: the partition denoted as a green $\times$ is only favored by the Shannon entropy, and the partition denoted as a yellow triangle is only favored by the min-entropy.

8 Related work

There is a long history of attacks that exploit visible patterns in encrypted web traffic. The first attacks for extract-



(a) Shannon entropy (b) Min-entropy

Figure 7. Auto-complete: the remaining uncertainty about the typed word in the auto-complete input field versus the relative expected overhead, for 500 randomly chosen initial parititions, using Shannon entropy and min-entropy as a measure. Red stars represent favored resulting partitions, i.e. resulting in a better security-versus-overhead trade-off than close-by points. The green $\times$ denotes a partition that is only favored by the Shannon entropy; the yellow triangle denotes a partition that is only favored by the min-entropy.

ing user information from the volume of encrypted web traffic were proposed in by Cheng and Avnur [8]; since then there have been several published attacks of this kind, e.g. [4, 7, 10, 12, 17, 22, 23, 30, 32, 40]. There has been an evolution of the goals which such attacks pursue, and the of settings of those attacks. While the goal of the attack presented in [8] is to identify a webpage within a website accessed through HTTPS, later attacks aim at identifying the website a user is visiting when browsing through an encrypted tunnel. Those attacks target HTTPS connections to anonymizing proxies [12, 23, 40], SSH or a VPN connections to anonymizing proxies [4, 22, 30, 32], data contained in anonymized NetFlow records [10], or onion routing and web mixes [22, 37]. Chen et al. [7] turn the focus of their attacks to web applications accessed through HTTPS and WPA, and the goal of those attacks is the extraction of sensitive information about user's health conditions and financial status.

Several works propose countermeasures against those attacks. *Padding*, i.e., adding noise to observations, is the standard countermeasure and has been proposed by a large number of authors [7, 8, 12, 23, 30, 32, 40]. A different approach proposed by [40] and implemented by [41] changes patterns of observations corresponding to one website to look like patterns corresponding to another website, which allows sensitive traffic to be camouflaged as traffic coming from popular services. Features of the TCP and HTTP protocols were utilized in the techniques proposed by Luo et al. [33] used to build the HTTPOS system which offers browser-side protection from traffic analysis. We have shown how parts of HTTPOS and other previously proposed countermeasures can be cast in our model, and how we can use it to formally reason about them.

A number of works empirically evaluate the security of proposed countermeasures [8, 30, 33, 37, 40, 41]. There, the security is measured by comparing the performance of attacks before and after the application of the countermeasures. In recent work, Dyer et al. [17] exhibit shortcomings in this kind of security evaluation. In particular, they demonstrate the inability of nine previously known countermeasures to mitigate information leaks, even when only coarse traffic features are exploited, such as total bandwidth or total time. In contrast, we propose a formal framework which does not assume the use of particular attacks, but rather measures security in terms of the amount of information leaked from the produced observations.

Sidebuster [42] is a language-based approach to countering side-channel attacks in web applications. It uses a combination of taint-based analysis and repeated sampling to estimate the information revealed through traffic patterns, however without an explicitly defined system model. Our goal for future work is to use our model for providing a semantic basis for the security guarantees derived by such language-based approaches. Liu et al. [31] present a traffic padding technique that is backed up by formal guarantees. In contrast to our work, their model does not account for an attacker's prior knowledge or for the probabilistic nature of traffic patterns. Finally, [5] propose a black-box web application crawling system that interacts with the web application as an actual user while logging network traffic. They view an attacker as a classifier and quantify information leaks in terms of classifier performance, using metrics based on entropy and on the Fisher criterion. Their metrics are computed from a small number of samples, which can

deliver imprecise results for entropy [3]. In contrast, we apply our metric to the entirety of paths, which is possible due to the random surfer assumption. Finally, work in quantitative information-flow analysis has been applied for assessing the information leakage of anonymity protocols [6] and cryptographic algorithms [27].

9 Conclusions and future work

We have presented a formal model that enables reasoning about side-channel attacks against web applications. In our model, the web application's encrypted traffic is cast as an information-theoretic channel, which gives a clean interface to the growing body of research in quantitative information-flow analysis and allows us to use established notions of quantitative security. We have demonstrated that our model is expressive enough to encompass web browsing, simple web applications, and several countermeasures from the recent literature. Furthermore, we have demonstrated algorithms allowing the efficient derivation of security guarantees for real systems.

Our long-term goal is to use the presented model as a semantic basis for language-based approaches for reasoning about side-channel attacks against web applications, such as [5,42]. Progress along these lines could lead to principled analysis techniques with soundness guarantees that hold for the whole protocol stack. As intermediate steps, we will investigate methods for approximating f_{net} by sampling, and we will investigate how quality guarantees for the sampling affect the final security guarantees. Moreover, we will investigate alternative notions of confidentiality [16] for scenarios in which we cannot justify assumptions about the uncertainty about the user's behavior.

Acknowledgments This work was partially funded by European projects FP7-256980 NESSoS, FP7-229599 AMAROUT, and Spanish project TIN2009-14599 DESAFIOS 10.

References

[1] M. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring Information Leakage using Generalized Gain Functions. In *Proc. IEEE Computer Security Foundations Symposium (CSF),*, pages 265–279. IEEE, 2012.

[2] A. Askarov, D. Zhang, and A. C. Myers. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, pages 297–307. ACM, 2010.

[3] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld. The complexity of approximating entropy. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 678–687. ACM, 2002.

[4] G. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Proc. Privacy Enhancing Technologies (PET)*, pages 1–11. Springer, 2005.

[5] P. Chapman and D. Evans. Automated black-box detection of side-channel vulnerabilities in web applications. In *Proc. 18th ACM Conference on Computer and Communications Security (CCS)*, pages 263–274. ACM, 2011.

[6] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. *Inf. Comput.*, 206(2-4):378–401, 2008.

[7] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy (SSP)*, pages 191–206. IEEE, 2010.

[8] H. Cheng, H. Cheng, and R. Avnur. Traffic analysis of ssl encrypted web browsing, 1998.

[9] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[10] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, and M. K. Reiter. On web browsing privacy in anonymized netflows. In *Proc. 16th USENIX Security Symposium*, pages 23:1–23:14. USENIX Association, 2007.

[11] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.

[12] G. Danezis. Traffic analysis of the http protocol over tls, 2007.

[13] S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal state-space lumping in markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.

[14] T. Dierks and C. Allen. The TLS protocol version 1.0, 1999.

[15] G. Doychev. Analysis and mitigation of information leaks in web applications. Master's thesis, Saarland University, Germany, 2012.

[16] C. Dwork. Differential Privacy. In *Proc. 33rd Intl. Colloquium on Automata, Languages and Programming (ICALP)*, pages 1–12. Springer, 2006.

[17] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-Boo, I still see you: Why Traffic Analysis Countermeasures Fail. In *IEEE Symposium on Security and Privacy (SSP)*, pages 332–346. IEEE, 2012.

[18] M. Eirinaki and M. Vazirgiannis. Usage-Based PageRank for Web Personalization. In *Proc. 5th IEEE Intl. Conference on Data Mining (ICDM)*, pages 130–137. IEEE, 2005.

[19] M. Eirinaki, M. Vazirgiannis, and D. Kapogiannis. Web path recommendations based on page ranking and markov models. In *Proc. 7th ACM Workshop on Web Information and Data Management (WIDM)*, pages 2–9. ACM, 2005.

[20] B. Espinoza and G. Smith. Min-entropy leakage of channels in cascade. In *8th Intl. Workshop on Formal Aspects in Security and Trust (FAST)*, pages 70–84. Springer, 2011.

[21] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

[22] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier. In *Proc. ACM Workshop on Cloud Computing Security (CCSW)*, pages 31–42. ACM, 2009.

[23] A. Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies (PET)*, 2002.

[24] D. C. Howe and H. Nissenbaum. TrackMeNot: Resisting surveillance in web search. In I. Kerr, V. Steeves, and C. Lucock, editors, *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, chapter 23, pages 417–436. Oxford University Press, Oxford, UK, 2009.

[25] J. Kemeny and J. Snell. *Finite Markov Chains*. Undergraduate Texts in Mathematics. Springer-Verlag, 1960.

[26] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proc. Annual Intl. Cryptology Conference (CRYPTO)*, pages 104–113. Springer, 1996.

[27] B. Köpf and D. Basin. An Information-Theoretic Model for Adaptive Side-Channel Attacks. In *Proc. ACM Conf. on Computer and Communications Security (CCS)*, pages 286–296. ACM, 2007.

[28] B. Köpf and M. Dürmuth. A Provably Secure and Efficient Countermeasure against Timing Attacks. In *Proc. IEEE Computer Security Foundations Symposium (CSF)*, pages 324–335. IEEE, 2009.

[29] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

[30] M. Liberatore and B. N. Levine. Inferring the Source of Encrypted HTTP Connections. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 255–263. ACM, 2006.

[31] W. M. Liu, L. Wang, K. Ren, P. Cheng, and M. Debbabi. k-indistinguishable traffic padding in web applications. In *Proc. Privacy Enhancing Technologies (PET)*, pages 79–99. Springer, 2012.

[32] L. Lu, E.-C. Chang, and M. C. Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Proc. 15th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2010.

[33] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, and R. K. C. Chang. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Proc. Network and Distributed Systems Symposium (NDSS)*. The Internet Society, 2011.

[34] J. L. Massey. Guessing and Entropy. In *Proc. IEEE Intl. Symp. on Information Theory (ISIT)*, page 204. IEEE, 1994.

[35] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[36] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[37] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proc. ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011.

[38] R. Sarukkai. Link prediction and path analysis using markov chains. *Computer Networks*, 33(1-6):377–386, 2000.

[39] G. Smith. On the foundations of quantitative information flow. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS '09)*, pages 288–302. Springer, 2009.

[40] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy (SSP)*, pages 19–30. IEEE, 2002.

[41] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proc. Network and Distributed Systems Symposium (NDSS)*. The Internet Society, 2009.

[42] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen. Sidebuster: automated detection and quantification of side-channel leaks in web application development. In *Proc. ACM Conference on Computer and Communication Security (CCS)*, pages 595–606. ACM, 2010.

A Implementing basic countermeasures

In the following, we discuss implementation considerations for each of the basic network fingerprints presented in Section 4. We distinguish between two kinds of fingerprints: We say that a fingerprint is deployed in a *protocol-independent* manner if the traffic is modified before entering the protocol stack, and in a *protocol-dependent* manner if the traffic is modified within the protocol stack.

Padding The countermeasure of up to 255 bytes padding of packets has been specified in the TLS protocol [14]. Padding can also be deployed independently of the protocol: On the server side, it can be implemented by augmenting web-objects with variable-length comments or special characters that will not be displayed by the browser, which is possible in most languages (HTML, JavaScript), and in file formats (JPEG, FLV). On the client-side, protocol dependent approaches are possible, e.g. by deploying the HTTP Range option, or by causing TCP retransmissions, see [33].

Dummy On the server, protocol-independent deployment of *dummy* as a countermeasure is possible e.g. by including hidden objects in an HTML file. On the client side, protocol-independent deployment is possible by requesting existing files from the server. For this, the client should know which files reside on the server. On the client side, a protocol-dependent approach for implementing *dummy* is to use HTTP range and TCP retransmission (see [33]).

Split On the server-side, a protocol-dependent deployment of *split* as a countermeasure is possible on the TCP layer, by segmenting a packet before it is sent on the network. On the client side, a protocol-dependent deployment is possible, e.g. by setting TCP advertising window, or by utilizing HTTP Range (see [33]). Splitting a file will incur an overhead because of additional packet headers and control messages.

Shuffle On the server side, deployment of *shuffle* is possible by preloading all web-objects in random order and using JavaScript to display the content of the HTML in ordered form, see [15] for an example implementation. On the server and client side, straightforward implementations are possible where all but the first (.html) file are requested in shuffled order.

k-parallelism On the client side, a protocol-independent deployment of k-par is possible, e.g. by requesting $k-1$ additional vertices for each visited vertex (similar to the functionality offered by [24]). On the server-side, a protocol-independent implementation is possible by including pages in hidden iframes.

Interleave To implement this countermeasure, it should become impossible to tell which files belong to which stream. A protocol-dependent approach to achieve this is by utilizing the HTTP pipelining option (see [33]), or by tunneling traffic through an encrypted proxy.