Fatemeh Rezaei* and Amir Houmansadr

TagIt: Tagging Network Flows using Blind Fingerprints

Abstract: Flow fingerprinting is a mechanism for linking obfuscated network flows at large scale. In this paper, we introduce the first *blind* flow fingerprinting system called TagIt. Our system works by modulating fingerprint signals into the timing patterns of network flows through slightly delaying packets into secret time intervals only known to the fingerprinting parties. We design TagIt to to enable reliable fingerprint extraction by legitimate fingerprinting parties despite natural network noise, but invisible to an adversary who does not possess the secret fingerprinting key. TagIt makes use of randomization to resist various detection attacks such as multi-flow attacks. We evaluate the performance and invisibility of TagIt through theoretical analysis as well as simulations and experimentation on live network flows.

Keywords: Traffic analysis, fingerprinting, anonymity systems

DOI 10.1515/popets-2017-0050

Received 2017-02-28; revised 2017-06-01; accepted 2017-06-02.

1 Introduction

Linking network flows is an important problem in various security and privacy applications. Particularly, it is widely known [19, 33] that linking network flows can be used by adversaries to compromise anonymity in Tor [8] and other anonymity systems [7, 17, 19, 20, 24, 40, 41, 45] by correlating the traffic patterns of ingress and egress flows. Alternatively, linking network flows has been suggested [5, 9, 17, 19, 20, 31, 38, 43, 44, 46] as a mechanism to trace back to cybercriminals who relay their traffic through previously compromised machines, known as stepping stone proxies [46].

***Corresponding Author: Fatemeh Rezaei:** University of Massachusetts Amherst, email: frezaei@cs.umass.edu
Amir Houmansadr: University of Massachusetts Amherst, email: amir@cs.umass.edu

Due to the wide use of encryption and similar content-obfuscation mechanisms, modern techniques for linking network flows solely rely on using *traffic patterns* that are not significantly impacted by encryption and network perturbations, such as packet timings and sizes, as opposed to packet contents or headers; such an analysis is broadly referred to as *traffic analysis* [2, 7, 33]. Traditionally, such traffic analysis for linking flows is performed *passively* [7, 9, 38, 43, 44, 46], i.e., by observing and correlating traffic patterns like packet timings and sizes. More recently, researchers have investigated an *active* type of traffic analysis for linking flows [17, 19, 20, 40, 41, 45]. In this approach, traffic patterns are slightly *perturbed* (e.g., by slightly delaying packets) in a way to embed an artificial pattern into network flows that can be used for linking related flows.

The majority of existing work on active traffic analysis is devoted to what is called *flow watermarking* [17, 20, 31, 41, 42, 45]. Recently, Houmansadr et al. [19] proposed an alternative type of active traffic analysis called *flow fingerprinting*. While flow watermarks aim at tagging flows with a single bit of information, flow fingerprints aim at tagging flows with several bits of information. This enables one to apply different tags on different flows, therefore perform a finer-grained traffic analysis as discussed by Houmansadr et al. [19]. Intuitively, designing flow fingerprints is more challenging than watermarks as they aim at embedding multiple bits of information. We further distinguish flow watermarks and fingerprints in Section 2.2.2. Figure 1 shows the setting of a flow fingerprinting mechanism.

Previous flow fingerprinting schemes [11, 19] are *non-blind*, i.e., the fingerprinters and fingerprint extractors need to establish a control channel to continuously communicate information about the flows they intercept. This is a significant obstacle to the scalability of such systems. In this paper, we design the first *blind* flow fingerprinting mechanism, which we call TagIt. Blind mechanisms are significantly more practical and scalable than non-blind mechanisms. In a fingerprinting scenario with n ingress and m egress flows (shown in Figure 1), a non-blind scheme requires $O(n)$ communication between the fingerprinting parties, compared to $O(1)$ in

blind mechanisms. Additionally, a non-blind mechanism imposes an $O(nm)$ computation overhead due to the pairwise-correlation of the flows observed by the fingerprinting parties. A blind scheme, however, imposes an $O(m)$ computation overhead since the extraction is performed on the individual egress flows observed by the extractor.

We pursue several goals in the design of our blind fingerprint, TagIt. First, flow fingerprints need to be imperceptible, otherwise they can be identified (and potentially removed) by an adversary. TagIt embeds fingerprints by slightly delaying network packets. TagIt also makes use of a secret fingerprinting key, shared between legitimate fingerprinters and extractors, such that detection or extraction of the embedded fingerprints is practically impossible to an adversary who does not know the key. Second, the embedded fingerprint signals should be resilient to natural network noise to enable reliable extraction by legitimate extractors. Towards this, we leverage error correcting codes to compensate for network noise, packet loss, and network delays.

We perform a mathematical analysis of TagIt's performance by modeling network flows and network noise using statistical models. Based on our analysis, we pick parameters that result in reliable fingerprinting performance. We demonstrate TagIt's performance through extensive simulations on real-world network traces. We also implement TagIt and test it on live network traffic.

The rest of this paper is organized as follows. We overview some background on traffic analysis in Section 2. In Section 3 we describe the design principals of the TagIt fingerprinting scheme. We describe TagIt fingerprinting system in Section 4, and describe TagIt extraction in Section 5. We statistically analyze TagIt in Section 6 by modeling the network flow behavior. We evaluate TagIt through simulations as well as experimentations in Sections 7 and 8. We finally discuss fingerprint invisibility and resilience to multi-flow attacks in Section 9.

2 Background and Related Work

We start by introducing the problem of linking network flows and overviewing previous work.

2.1 Linking Flows

The problem studied in this paper is to *link encrypted network flows when they pass through obfuscating proxies*. In particular, this problem has been extensively studied in two contexts. First, an adversary may aim at de-anonymizing connections made through an anonymization system like Tor by *linking* the ingress and egress flows observed at various vantage points controlled by the adversary. For instance, the adversary shown in Figure 1b can de-anonymize a Tor connection by linking the corresponding ingress and egress flows observed on compromised guard and exit relays (or the network routers intercepting those flows). Note that such linking can not be done by comparing packet contents due to anonymization and onion routing.

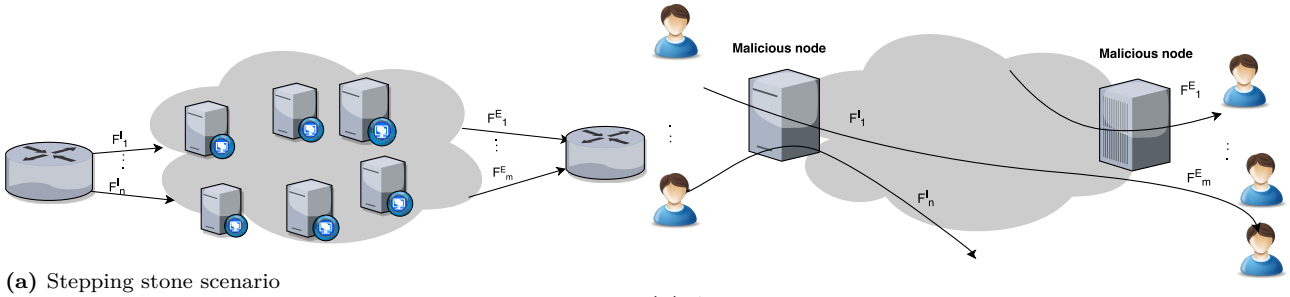
A second widely studied scenario for linking network flows is to identify stepping stone attackers [5, 9, 31, 38, 43, 44, 46]. A stepping stone attacker is one who relays her attack traffic through previously compromised machines, called stepping stone proxies. Figure 1a shows an example scenario. Similar to the anonymity scenario, the use of encryption by stepping stone proxies prevents linking flows through packet contents. Linking network flows has been studied in other scenarios as well, for instance, for detecting botmasters who control botnets trough low-latency, interactive C&C channels [18, 32].

2.2 Previous Work

In this section, we overview existing work on linking network flows using traffic analysis.

2.2.1 Passive Analysis

The traditional approach for linking network flows is mainly based on observing network traffic, and trying to link network flows by correlating their inherent characteristics such as packet timings, counts, and sizes [7, 9, 38, 43, 44, 46]. Zang and Paxson [46] model a network flow as a sequence of ON and OFF intervals, and correlate such ON/OFF patterns across flows. Alternatively, Blum et al. [5] correlate flows based on the number of packets received at any given point in time. They declare two flows to be correlated if their counts of packets are correlated over time. Such passive linking



(a) Stepping stone scenario

(b) Anonymity scenario

Fig. 1. Example application scenarios of network flow fingerprints. (F^I_* and F^E_* represent ingress and egress flows)

algorithms suffer from needing large numbers of packets before being able to make a reliable decision. Elices et al. [12] recently proposed a mechanism based on the Neyman-Pearson lemma that is able to link flows with fewer numbers of packets.

2.2.2 Active analysis

The major limitation of passive analysis is not being scalable to real-world applications. For a scenario with n ingress flows and m egress flows, passive analysis requires $O(n)$ communication and $O(nm)$ computation overheads, since the collected patterns need to be cross-correlated. Active traffic analysis reduces communication and computation overheads to $O(1)$ and $O(m)$, respectively, by embedding imperceptible tags into the flows being inspected. Most active analysis techniques [11, 17, 19, 20, 31, 41, 42, 45] work by modifying packet timings, i.e., by adding artificial delays to network packets to insert invisible timing tags.

There are two types of active analysis mechanisms.

Flow Watermarking. The main body of work on active flow linking is what is known as flow watermarking. In this approach, a traffic analysis party perturbs traffic patterns in such a way to reliably encode a *single* bit of information into network flows. Wang et al. [43] were the first to propose a flow watermarking mechanism by modulating the watermark signal into the inter-packet delays (IPDs). RAINBOW [20] also uses IPDs for watermarking but uses a *non-blind* architecture. It achieves a higher detection accuracy by using a side-channel to communicate the observed packet timings among traffic analysis parties. An alternative approach to flow watermarking is the interval-based approach in which packets are delayed into specific time intervals for watermark insertion. Most of the recent proposals for watermarking have used an interval based design [31, 41, 45] due to its better resistance to packet perturbations, like packet re-ordering and drops, compared to the IPD-based approach. For instance, Yu et al. [45] use spread spectrum pseudo-noise codes to modulate the watermark signal into the rate of packets in specific time intervals.

Early interval-based watermarks are susceptible to a multi-flow attack (MFA) [22], which works by aggregating multiple flows watermarked using the same key. Houmansadr et al. design Swirl [17] to be resistant to MFA by making the watermarking process dependent on the network flows being watermarked.

Flow Fingerprinting. Flow fingerprinting aims at embedding *multiple* bits of information on each network flow, as opposed to a single bit of information in flow watermarking. This enables the use of flow fingerprints in scenarios with significantly larger scales than that of watermarks. For instance, while an anonymity adversary (Figure 1b) can use a watermark to de-anonymize a single target connection, she can use fingerprints to de-anonymize a large number of connections (e.g., by embedding distinct fingerprint tags on different ingress flows she is intercepting). Since fingerprints need to embed multiple bits of information, the design of reliable fingerprinting systems is significantly more challenging than watermarking systems. We refer the reader to Houmansadr et al. [19] for detailed comparison of flow watermarking and fingerprinting.

Fancy [19] is the first flow fingerprinting mechanism. It extends the Rainbow [20] watermarking system through the use of various coding mechanisms in order to enable reliable insertion of multiple bits of fingerprints. Fancy is a—non-blind—fingerprint, i.e., the fingerprinting entities need to constantly communicate the information about the network flows they intercept through a side-channel. Non-blind fingerprinting (also non-blind

watermarking) suffers from similar scalability issues of passive traffic analysis mechanisms, i.e., an $O(n)$ communication overhead and an $O(nm)$ computation overhead. This is because the non-blind fingerprinting entities need to continuously communicate some information about the flows they intercept. In Fancy, for instance, the fingerprinters send the IPDs of the flows they have fingerprinted to the fingerprint extraction entities. Elices et al. [11] use a game-theoretic analysis to identify optimal strategies for non-blind fingerprinting and compare their performance with Fancy.

In this paper, we design—the first—blind fingerprinting system, TagIt. Similar to blind watermarks, a blind fingerprinting system reduces the communication and computation overheads to $O(1)$ and $O(m)$, respectively, compared to the $O(n)$ communication and $O(nm)$ computation overheads of non-blind mechanisms.

2.3 Relevance to Covert Communications

Active traffic analysis, including both flow watermarking and fingerprinting, can be considered as a specific type of covert communications, tailored to the application of linking flows. That is, one can consider the fingerprinter and extractor entities (Section 2.4) as covert communicating parties.

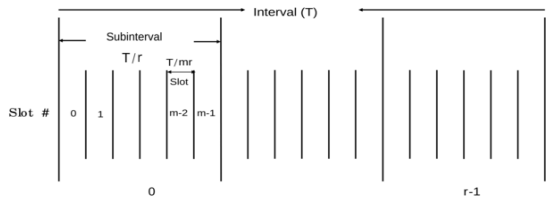
There is a number of works on covert communications that, similar to our active traffic analysis, try to encode information into the timings of network packets. However, such mechanisms are not applicable to the problem of linking flows studied in this paper. Particularly, a significant number of covert traffic timing mechanisms work by generating *synthetic* network flows—as opposed to modifying some existing flows—in order embed covert messages [1, 6, 15, 23, 27, 28, 39]. The synthetic traffic is generated in a way to mimic that of normal traffic. Such mechanisms are *not* usable for the application of linking flows, as introduced in this paper, since the covert communicating parties in our threat model do not generate traffic, but only perturb some existing traffic. Another class of covert timing channels work by injecting packets carrying covert messages into existing overt traffic [3, 10, 36, 37]. Such mechanisms also can not be applied to our problem, e.g., an adversary can not inject packets into an intercepted Tor connection.

There are some covert mechanisms that, similar to active traffic analysis, embed covert messages by perturbing the timings of overt traffic. Such proposals, however, are mainly inapplicable to our problem of linking network flows for various reasons. For instance, some of such techniques offer very low covert capacities [34, 35]; some are fragile to natural packet perturbations as they directly modify inter-packet delays [16]; and some require out-of-bound channels to constantly communicate traffic models [15] or traffic features [16]. Active traffic analysis, as studied in this paper, is a particular subclass of covert communication tailored to the specific problem of linking flows.

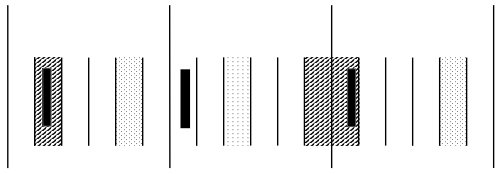
2.4 Threat Model

A flow fingerprinting system consists of two main components: *fingerprinters* and *extractors*. A flow fingerprinter is the party who manipulates network flows in order to embed fingerprint tags on those flows. On the other hand, the fingerprint extractor is the entity who inspects network flows to detect those with fingerprints, and to extract the fingerprints. For instance, in the anonymity scenario of Figure 1b the fingerprinter can be a malicious guard relay (or the ISP hosting a Tor guard relay), and the extractor can be an accomplice exit relay. On the other hand, in the stepping stone scenario of Figure 1a the fingerprinter and extractor can be the border routers of an enterprise network aiming at detecting stepping stone attacks through fingerprinting egress flows.

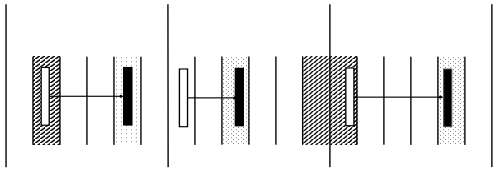
A fingerprinting *adversary* is an entity who aims at detecting the presence of fingerprints (and therefore, take actions to evade or destroy fingerprints). For instance, in the anonymity scenario of Figure 1b the fingerprinting adversary could be the communicating parties who use Tor, or the Tor project operators. On the other hand, in the stepping stone scenario of Figure 1a the fingerprinting adversary could be the cybercriminals who aim at detecting and evading potential flow fingerprinting systems. To ensure fingerprint invisibility against the fingerprint adversaries, the fingerprinters and extractors secretly share a *fingerprinting key* in advance. Ideally, no entity should be able to detect (or extract) a fingerprint unless she has access to this fingerprinting key (i.e., only legitimate fingerprint extractors).



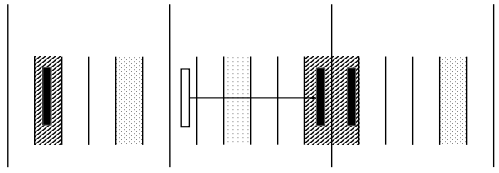
(a) A TagIt time interval



(b) Original flow



(c) Embedding bit 0



(d) Embedding bit 1

Fig. 2. Embedding bits 0 and 1 by TagIt. To embed 1, packets should move to darker subintervals, and to embed 0, they should be moved to lighter subintervals.

3 TagIt: Design Principles

We first overview the main principles in the design of the TagIt fingerprinting system.

Timing-based fingerprinting: Similar to the large fraction of previous works on active traffic analysis, TagIt is a timing-based mechanism. Therefore, a TagIt fingerprinter embeds fingerprints into an intercepted flow by modifying the timings of its packets, i.e., by delaying some of its packets. This makes the fingerprints usable in scenarios where the content of the flows being linked is modified in transit by obfuscating proxies.

Blind design: TagIt is designed to be a blind fingerprinting system; that is, the *only* information shared between TagIt's fingerprinter(s) and extractor(s) is a fingerprinting key. This is unlike existing "non-blind" flow correlation designs [11, 19, 20] where they need to additionally share some information about the intercepted flows. Particularly, in the Fancy non-blind fingerprint [19] the fingerprinter will need to continuously send the inter-packet timings of all the flows it intercepts to the Fancy extractors for the extraction mechanism to work.

Blind designs are significantly more practical in real-world applications. As discussed above in Section 2.2.2, a blind traffic analysis system offers an $O(1)$ communication and $O(m)$ computation in a scenario with n ingress and m egress flows being intercepted. This is while the orders of communication and computation are $O(n)$ and $O(nm)$, respectively, for a non-blind system like Fancy. Note that when comparing the orders of computation between two fingerprinting systems, one should also factor in the computation overhead of every single correlation operation, as this may differ across different systems (e.g., due to their use of different coding algorithms). This is not included in our computation order analysis, since the same correlation algorithms used in a blind system could be also used by a non-blind system.

Interval-based: There are two types of timing-based active traffic analysis systems: interval-based and IPD-based systems. An IP-based system encodes the fingerprint signal into the inter-packet delays of packets, i.e., by modifying IPDs individually. On the other hand, an interval-based approach modulates the fingerprint signal into the counts of packets that arrive within specific time intervals. We use an interval-based structure for TagIt. This is because interval-based systems offer significantly stronger resistance to natural packet modifications, such as packet drops, repacketization,

packet reordering, etc., compared to the IPD-based systems, known [19–21] to be susceptible to such modifications. Therefore, a TagIt fingerprinter delays packets into specific time intervals in order to fingerprint a flow, which we call *fingerprint* intervals. A TagIt extractor will count the ratio of the packets arriving in such intervals in order to perform fingerprint extraction.

Randomized insertion: TagIt's interval-based approach makes it robust to packet-level perturbations, as discussed above. However, Kiyavash et al. [22] demonstrate that interval-based schemes may be susceptible to an attack called the multi-flow attack (MFA). In this attack, the adversary collects multiple flows fingerprinted using an interval-based mechanism, and superimposes the flows to increase the chances of identifying the fingerprint intervals. The attack is built on the statistical distribution of packets in various intervals of an interval-based scheme.

We design TagIt in a way to be resistant the MFA attack despite its interval-based scheme. Specifically, TagIt uses a random mechanism for selecting the fingerprinting intervals in a way that even inserting *the same fingerprint* into *the same flow* twice will result in different fingerprinted flows. As we will show in our analysis of Section 9, this makes TagIt resist the MFA by smoothing the statistical distribution of the packets in the aggregation of multiple TagIt flows.

Coding to resist noise: As described earlier, designing a reliable fingerprinting system is significantly more challenging than a watermarking system since a fingerprinting system aims for the reliable transmission of multiple bits of information across a noisy communication channel, unlike watermarking's single bit of information. We use two stages of coding to enable reliable fingerprint extraction despite the network noise. First, we use a repetition code to resist noise due to normal network jitter. Second, we make use of standard error correcting codes (like Convolutional or Reed-Solomon codes) to resist sparse, bursty channel errors that are not recoverable by normal repetition codes. We will further discuss the specific choices of our coding parameters. As will be shown in our analysis of Section 8, such codings result in a promising reliability for TagIt's fingerprint extraction.

4 TagIt Fingerprinting Scheme

In this section, we discuss the algorithm used by a TagIt fingerprinter to fingerprint network flows. As discussed earlier, TagIt is a timing-based, interval-based scheme, therefore, it works by delaying some of the packets of a flow to be fingerprinted into specific timing intervals. In the following, we describe the details of TagIt fingerprinter; Figure 2 illustrates TagIt's fingerprinting process.

Dividing a flow into time intervals. The fingerprinter divides the time axis into a series of consecutive time intervals of lengths T with the first interval starting at the time *offset* $0 \leq o < T$. That is, the ith interval includes the packets arriving during the time period $[o + (i-1)T, o + iT]$. The fingerprinter uses the arrival time of the first packet in the candidate flow as time zero.

Selecting Fingerprint Intervals. The TagIt fingerprinter embeds a flow fingerprint by delaying packets within several time intervals of that flow, which we call them *fingerprint intervals*. Suppose that the fingerprinter aims at inserting an ℓ-bits long fingerprint tag into a candidate flow. The fingerprinter converts the ℓ-bits fingerprint into $\ell^c = f/r_c$ bits of *encoded fingerprint* bits using a Convolutional or Reed-Solomon encoder, where r_c is the coding rate of our encoder and its choice will be discussed later in Section 8. The fingerprinter uses the first n time intervals of a flow as its fingerprint intervals, and assigns the ℓ^c encoded fingerprint bits to these intervals in order (as discussed later, TagIt may insert multiple bits in each interval).

Dividing fingerprint intervals into slots. The fingerprinter divides each fingerprint interval (of length T) into r sub-intervals of equal length T/r. It further divides every sub-interval into m slots of length $T/(rm)$. We refer to the jth slot of the ith sub-interval as $s_{i,j}$ $(i = 1, .., r;\ j = 1, .., m)$. Figure 2a shows a fingerprint interval.

Secret permutation functions. A fingerprinter generates two vectors of permutation functions, $\Pi^0 = (\pi_0^0, \pi_1^0, ..., \pi_{r-1}^0)$ and $\Pi^1 = (\pi_0^1, \pi_1^1, ..., \pi_{r-1}^1)$, before starting the fingerprinting process, and shares them secretly with the extractor(s) as part of the secret fingerprinting key. Each π_j^i is a permutation function on $\mathbb{Z}_m = [0, .., m-1]$; for instance, $\pi_j^i : [0, 1, 2, 3, 4, 5] \rightarrow [3, 5, 1, 0, 2, 4]$ is an example permutation function for $m = 6$.

Note that the Π^0 and Π^1 functions are generated once and used for fingerprinting many flows. In Section 4.2 we will discuss the implications of the random selection of Π^0 and Π^1.

Selecting fingerprint slots. For each fingerprint interval, the fingerprinter uses the permutation functions Π^0 and Π^1 along with a *seed* to select its *fingerprint slots*. That is, for the kth fingerprint interval, the fingerprinter selects the following set of slots as the fingerprinting slots:

$$Z_k = (\pi_0^b(d_k), \pi_1^b(d_k), ..., \pi_{r-1}^b(d_k)) \quad (1)$$

where $b \in \{0, 1\}$ is the encoded fingerprint bit to be embedded in the kth interval, and d_k is a random seed (described later). Note that to improve resistance to exhaustive key search attacks, we use different Π^0 and Π^1 functions for different fingerprint intervals (this is analyzed in Section 9.4).

Embedding a fingerprint bit. Finally, the fingerprint bits are embedded by the fingerprinter delaying packets into the fingerprint slots. That is, the kth encoded fingerprint bit is embedded into the kth interval by delaying that interval's packets into their nearest fingerprint slots in Z_k (the packets are delayed forward, so packets appearing after an interval's last fingerprint slot are not delayed). This illustrated in Figure 2.

As we will analyze in Section 9, for high-rate flows delaying all packets into fingerprint slots will weaken the invisibility property. We therefore only delay a fraction R_{move} of the packets into the fingerprint slots. Suppose that Δ is the length between two consecutive fingerprint slots; the fingerprinter only delays the packets arriving in the last $\Delta \times R_{move}$ part of the inter-slot interval into the second fingerprint slot. This is shown in Figure 3. We discuss this parameter in Section 8.

Empty intervals For the intervals that we have no packet to fingerprint, we simply ignore that interval, and therefore lose the bit corresponding to that interval. Note that our choice of parameters are so that such empty intervals are rare. Also, our use of coding compensates for some of such lost bits. Alternatively, one could use the next non-empty interval to embed the corresponding bit; however, this will increase the risk of de-synchronization between the extractor and fingerprinter, e.g., if a single packet moves into an empty interval all of its following bits will be lost at the extractor.

Secret key of fingerprinting. Table 1 summarizes the parameters of our system. Also, Table 1 shows the parameters that are part of the fingerprinting key. The key parameters are secretly shared between the fingerprinter and extractor, while the other parameters may be publicly disclosed.

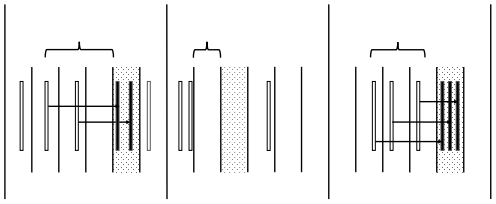


Fig. 3. To ensure invisibility, TagIt fingerprinter only moves R_{move} fraction of packets into fingerprint slots. R_{move} depends on the rate of the flow being fingerprinted.

Table 1. Fingerprint Parameters

System parameters	
T	Interval length
r	Number of subintervals
m	Number of slots per subinterval
n	Number of intervals
τ	Packet extraction threshold
ρ	Fingerprint extraction threshold
ℓ	Fingerprint length
m_ℓ	Slot length
n_{bit}	Num. of fingerprint bits embedded in each interval
R_{move}	Fraction of fingerprinted packets
Secret parameters	
Π^0	Permutation for embedding bit 0
Π^1	Permutation for embedding bit 1

4.1 Extension: Inserting multiple bits per interval

As discussed above, TagIt uses Π_0 and Π_1 to embed bits 0 and 1, respectively. We extend the set of permutation functions to $\Pi_0, \Pi_1, ...\Pi_{S-1}$ in order to be able to embed S different symbols. In other words, TagIt can embed $n_{bit} = \log_2 S$ bits of information per interval by using S numbers of permutation functions.

As expected, using more permutation functions increases the extraction complexity as the extractor will need to check more slot mappings. It also increases the risk of extraction errors: as we increase the number of

permutation functions, the probability of their overlap also increases, which results in extraction errors. This can be seen in Figure 4a, where increasing the number of bits inserted per interval also increases the probability of error. Figure 4b shows the average number of bits reliably embedded per second for various numbers of permutation functions.

4.2 Analysis of Permutation Functions

Each permutation function gives the fingerprinter m *slot mapping*s for fingerprint insertion (e.g., $\Pi^0(d)$ gives m slot mappings for bit 0 for a seed value d). However, since the permutation functions are generated randomly, it is likely that two slot mappings where one represents a 0 bit and the other represents a 1 bit have overlapping slots. A high fraction of such overlapped slots between two mappings that represent different bits will increase the rate of false extractions, i.e., a 0 bit encoded into an interval may be decoded as 1. Therefore, it is important to evaluate such overlaps for random selections of $\Pi^0(d)$.

Let us compute the expected number of overlaps between two permutation functions. The probability of having $i \in [0, 1, \cdots, m-1]$ overlaps between two randomly generated permutation functions is given by

$$P_i = \binom{N}{i} \sum_{j=0}^{N-i} (-1)^j (N-i-j)!/N! \qquad (2)$$

where $N = m!$ is the possible number of permutation functions. Therefore, we derive the expected fraction of overlaps between two permutation functions as

$$R_{overlap} = \frac{1}{m} \sum_{i=0}^{m} iP_i \qquad (3)$$

Therefore, for $m = 6$ (used in our experiments) we have $R_{overlap} = 0.166 \simeq 1/6$. We confirm this analysis by randomly generating 40 vectors of permutation functions each of length 6000 and measuring their overlaps. As can be seen in Figure 5 the mean probability of overlap is close to 0.166.

5 Fingerprint Extraction

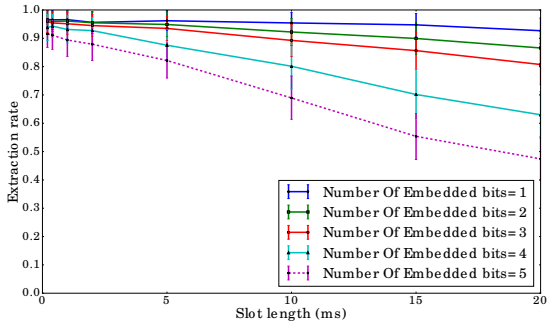
In this section, we describe how a TagIt extractor can extract fingerprints from the flows it intercepts. For a network flow intercepted by a TagIt extractor, it will either declare the flow as "not fingerprinted", or will extract a fingerprint tag from that flow.

The extractor knows the fingerprinting key used by fingerprinters. As described above, the fingerprinter uses a random seed $d_k \in \mathbb{Z}_m$ to select a slot mapping for the kth interval. Therefore, the extractor will need to use $M = 2^{n_{bit}} \cdot m$ possible mappings, e.g., m mappings for the bit 0 and m for the bit 1 when $n_{bit} = 1$. For each of the possible M slot mappings, the extractor evaluates the ratio of the packets appearing in those mappings. For instance, $R^I(k)$ is the fraction of packets in interval I that appear in the slots according to the kth mapping ($0 \le k \le M-1$). Finally, the extractor finds the mapping k_{max} that has the maximum fraction, i.e., $k_{max} = \arg\max_{R^I(k)} k$.

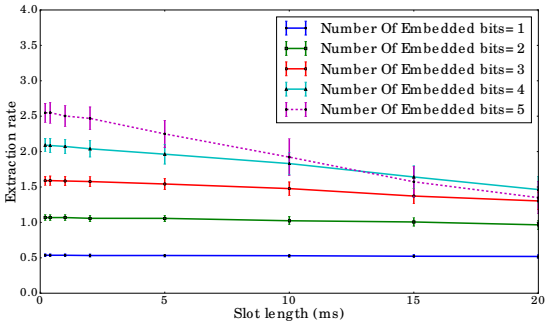
If $R^I(k_{max}) > \tau$, the extractor declares the extracted bit to be the bit represented by the mapping k_{max}. Otherwise, the extractor extracts no bit from the interval I, i.e., returns a *null* bit. Note that τ is a parameter of the extractor and makes a trade-off between the false positive and negative rates. For the extraction to be successful, we need to have $\tau \le R_{move}$, where R_{move} is the fraction of packets moved into fingerprint slots, as defined earlier. The final stage of extraction is to use coding to correct the potentially corrupted bits. The goal of the fingerprinter is to be able to extract all fingerprint bits from a fingerprinted flow, or to correctly declare a non-fingerprinted flow as non-fingerprinted.

Extraction complexity. For each fingerprint interval, the legitimate extractor who knows the fingerprinting key, i.e., the fingerprint mappings, will need to count only M possible mappings (e.g., $2m$ for $n_{bits} = 1$). An adversary will need to guess the fingerprinting key to be able to detect the fingerprint; as we show in Section 9.4 TagIt's key has an extremely high entropy. For each key, the adversary needs to try the possible M mappings to see if any of them decode a fingerprint bit.

Another factor in the complexity of the extraction process is the complexity of the coding scheme used by the fingerprinters. However, note that since an adversary should also perform the same decoding algorithms, the coding complexity applies to the adversary as well. The coding complexity of TagIt will depend on each specific coding scheme used. For instance, a convolutional code can be decoded with a low-complexity Viterbi decoder [13, 14, 26, 29], which performs only 2^{kL} checks for each decoding operation. (Please see Appendix A for



(a) Extraction rate for various number of bits per interval



(b) Average number of bits reliably extracted per second

Fig. 4. Inserting multiple bits per interval.

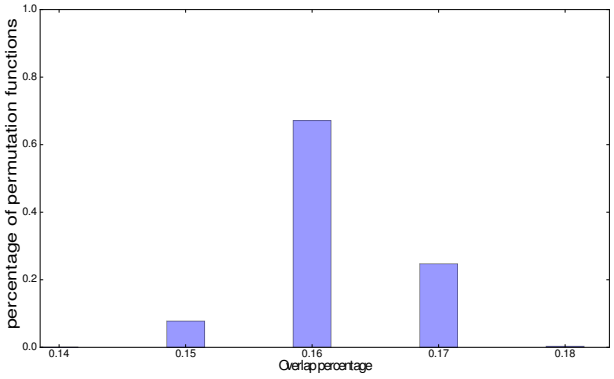


Fig. 5. Empirical distribution of slot overlaps between randomly generated permutation functions with length 6000.

parameters. This is 2^3 operations for our 1/3 convolutional encoder with a constraint length of 3.)

Extraction Synchronization. To extract the correct fingerprint bit, we have to synchronize the received flow with the sent one. To do so, we try multiple offset values in the range $[0, T/r]$ using steps of length t, i.e., $T/(rt)$ computations. We experimentally find that setting $t = m_\ell/4$ makes the right balance between computation and synchronization accuracy. Figure 6 shows an extractor trying various offset values.

6 System Analysis

6.1 False extraction of fingerprints from non-fingerprinted flows

We evaluate the probability of extracting a fingerprint message from a non-fingerprinted flow. In order to extract a fingerprint bit from the fingerprint interval of a non-fingerprinted flow, a fraction τ or larger of the packets within that interval should arrive within the fingerprint slots of some fingerprint mapping.

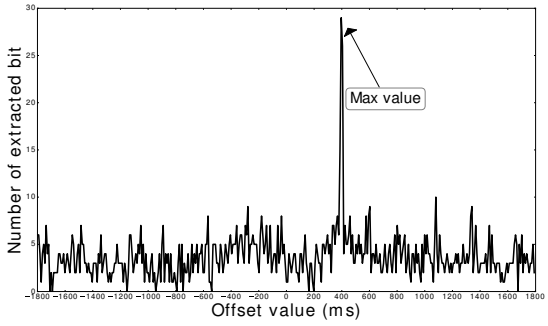
For a particular fingerprint interval, consider a specific fingerprint mapping (out of M possible mappings). For that mapping, the interval is declared to be fingerprinted with the corresponding bit if at least τ fraction of the packets in that interval arrive within the fingerprint slots of that mapping. We assume the packets to have a Poisson distribution, i.e., the inter-arrival times are i.i.d. Therefore, for each packet, it will arrive within a fingerprint slot with a $\frac{1}{m}$ probability. Therefore, assuming P total packets in a fingerprint interval, the chances of having $\tau \times P$ or more packets within the fingerprint slots is given by:

$$F_P = 1 - CDF_{P,1/m}^{Binomial}(\lceil \tau.P \rceil) \tag{4}$$

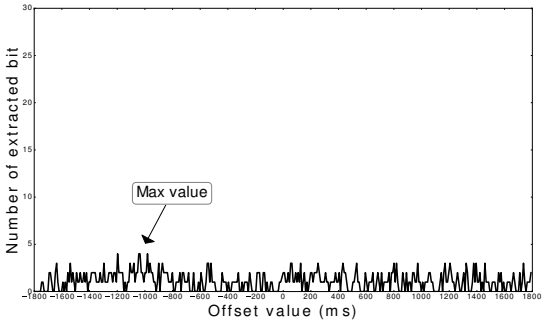
where $CDF_{n,p}^{Binomial}(x)$ is the CDF function of a Binomial distribution with n number of trials and p probability of success. We have that

$$CDF_{n,p}^{Binomial}(x) = \sum_{i=0}^{x} \binom{n}{i} p^i (1-p)^{n-i}$$

We model the arrival times using a Poisson process with rate λ (λ is the flow rate). Consequently, the number



(a) Fingerprinted flow



(b) Non-fingerprinted flow

Fig. 6. Offset synchronization of fingerprint extraction

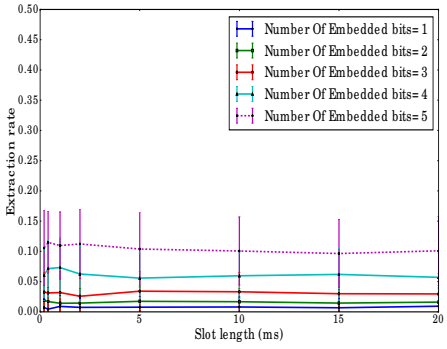


Fig. 7. False positive according to the number of embedded bits in interval. Each point in the figure is the average over 100 flows.

of packets in the interval follows a Poisson distribution with mean $\lambda.T$ (T is the interval length). Therefore, we average F_P for different values of P as follows:

$$F_I = M \sum_{P=1}^{\infty} \frac{e^{-\lambda.T}(\lambda.T)^P}{P!} F_P \tag{5}$$

Note that we also multiplied F_P with M, the number of all possible fingerprint mappings.

F_I is the probability of extracting a fingerprint bit from an arbitrary (non-fingerprinted) interval. Therefore, the chances of extracting ρ fraction of all fingerprint bits (out of total n fingerprint intervals) follows a Binomial distribution, which gives us the false positive extraction rate:

$$FP = 1 - CDF_{n,F_I}^{Binomial}(\lceil \rho.F_I \rceil) = \sum_{k=n\rho}^{n} (n,k)(F_I)^k(1-F_I)^{n-k} \tag{6}$$

6.2 Fingerprint error rates

Remember that for a given fingerprint interval, the fingerprinter moves R_{move} fraction of its packets to the fingerprint slots corresponding to a particular mapping. For the extractor to be able to extract this fingerprint bit, at least τ fraction of packets should be still in the corresponding fingerprint slots. Therefore, an error happens when more than $R_{move} - \tau$ fraction of packets within the interval move out of the fingerprint slots.

An individual packet moving out of a fingerprint slot. Consider a fingerprinted packet p_i, i.e., one that has been moved to a fingerprint slot by the fingerprinter. Suppose that p_i has a distance x from the center of its fingerprint slot ($-\frac{T}{2mr} \leq x \leq \frac{T}{2mr}$). We model network noise on packets with a Gaussian distribution as suggested in previous work [20, 30] and also confirmed in our measurements shown in Figure 8. Therefore, the probability of p_i moving out of its slot due to noise is:

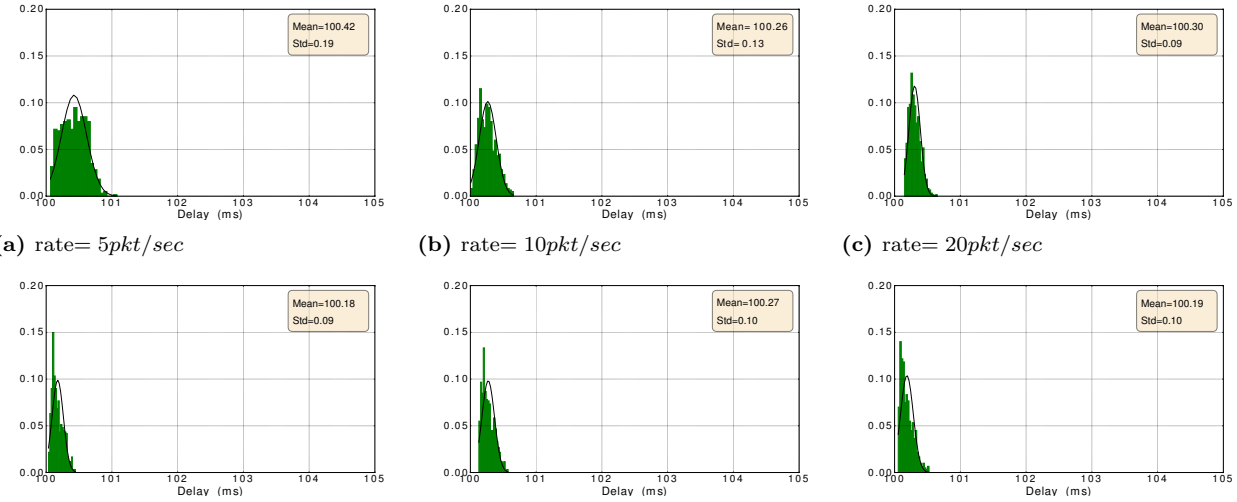
$$P(p_i \text{ shifted}|x) = 1 - \Phi_{0,1}(\frac{T/2mr - x}{\sigma}) + \Phi_{0,1}(-\frac{T/2mr + x}{\sigma}) \tag{7}$$

where $\Phi_{0,1}(\cdot)$ is the CDF of a Gaussian distribution with mean 0 and standard deviation 1.

Given p_i's uniform distribution in the fingerprint slot, we have:

$$P(p_i \text{ shifted}) = \frac{rm}{T} \int_{x=-T/2mr}^{x=T/2mr} P(p_i \text{ shifted}|x)dx \tag{8}$$

Losing a fingerprint bit. The fingerprinter loses an interval's fingerprint bit if more than $R_{move} - \tau$ fraction



(a) rate= $5pkt/sec$ **(b)** rate= $10pkt/sec$ **(c)** rate= $20pkt/sec$

(d) rate= $40pkt/sec$ **(e)** rate= $100pkt/sec$ **(f)** rate= $200pkt/sec$

Fig. 8. Delay distribution for different rates on a link between two distant computers (one in the U.S. and the other in Europe).

of its packets move out of the designated fingerprint slots, i.e.,

$$Pr_{loss} = Pr\{(R_{move} - \tau) \text{ or more frac. of packets shifted}\}$$

Assuming there are P packets in the interval, we have

$$Pr^{P}_{loss} = 1 - CDF^{Binomial}_{P,c}(\lceil (R_{move} - \tau)P \rceil) \qquad (9)$$

where $c = P(p_i \text{ shifted})$ in (8). By averaging across all possible P's we have

$$Pr_{loss} = \sum_{P=1}^{\infty} \frac{e^{-\lambda.T}(\lambda.T)^P}{P!} Pr^{P}_{loss} \qquad (10)$$

which is the probability of the extractor losing one particular fingerprint bit. Note that this is an upper bound for the error; for each bit there are multiple mappings (e.g., $M/2$ for $n_{bit} = 1$), and therefore the packets of the noisy interval may move into slots that correspond to another mapping of the fingerprinted bits.

Flipping a fingerprint bit. A lost fingerprint bit may be decoded as an invalid bit, e.g., a 0 bit may be extracted as a 1 by the extractor. This happens if the shifted packets move to the slots corresponding to a mapping that represents an incorrect bit. We can estimate the probability of a bit flip as:

$$Pr_{flip} = Pr_{loss} \times (M_b \sum_{P=1}^{\infty} \frac{e^{-\lambda.T}(\lambda.T)^P}{P!} F_P) \qquad (11)$$

where F_P is given in (4) and M_b is the number of possible mappings for the incorrect bit.

Extraction error rate. Recall that TagIt uses an encoder, e.g., a convolutional code, to transform ℓ fingerprint bits into ℓ^c coded bits. Suppose that our encoder can correct c bits out of ℓ^c bits. Therefore, our extractor should be able to correctly extract $\ell^c - c$ or more in order to be able to decode the ℓ fingerprint bits.

Therefore the probability of the extraction error is:

$$Error_{Extraction} = 1 - CDF^{Binomial}_{\ell^c, Pr_{loss}}(c) \qquad (12)$$

As discussed in Section 8, we choose our parameters such that $Error_{Extraction}$ is close to zero.

7 Simulations

7.1 Simulation setup

In our simulations, we generate synthesized network flows with various rates based on Poisson processes, as commonly used in the literature. We measure network jitter between a node on our campus (which we will call "Campus") and several Planetlab nodes [4]. We particularly pick three Planetlab nodes that represent various network conditions. Table 2 compares the standard deviation of delay for the three links that we used in our simulations. The delays are measured over 200 flows each containing 600 packets sent over the links.

We additionally simulate more noisy conditions (higher delay standard deviations) by adding artificial noise to

Table 2. Comparing the network delay of the three links used in our simulations.

	Nodes involved	Standard deviation of delay
Link 1	Campus-Ireland	$0.083 - 0.478$ (ms)
Link 2	Campus-Switzerland	$9.773 - 12.347$ (ms)
Link 3	Campus-China	$24.917 - 30.762$ (ms)

Table 3. Trade-offs in selecting different parameters of TagIt

Parameter	Trade-offs	
	Increasing improves	Decreasing improves
T	Extraction rate	Extraction time
r	Delay, invisibility	Extraction rate
m_ℓ	Extraction rate	Invisibility
m	Invisibility, delay	Extraction rate
n_{bit}	Invisibility	False pos., Extract. time
τ	False positive	False negative
ρ	False positive	False negative
ℓ	Extract. performance	Extraction time, delay
R_{move}	Extraction rate	Invisibility

the traffic using the Linux `tc` command. For each link, we measure delays for three different flow rates of 10, 100, and 200 pkt/sec.

Note that we have excluded experiments on Tor. This is because our measurement of delays on Tor suggests that they significantly deviate from that of typical Internet traffic, therefore our earlier analysis does not hold. We leave deriving parameters tailored to Tor for future work.

7.2 Parameter choices

Table 3 shows how each of TagIt's parameters impact its performance. For instance, r makes a trade-off between false-negative and the added delay, since the maximum inserted delay is bounded by $\frac{T}{r}(2 - \frac{2}{m})$.

We pick $m = 6$ throughout the paper. As mentioned, m makes a trade off between extraction performance and invisibility. The parameter T should be chosen based on the rate of the flow, since false errors are proportional to $T\lambda$. The parameter m_ℓ represents a trade-off between extraction time and delay since having small length for m_ℓ will make the synchronization step more time-consuming as discussed in Section 5. τ should be used to control the rates of false positive and false negative. Increasing τ improves the false positive, and decreasing it improves the false negative. Also, ρ represents a trade-off between false positive and false negative rates. Figure 7 suggests a maximum false extraction rate of 0.15, i.e., an expected number of $30 * 0.15 = 4.5$ false bits when $n = 30$. We therefore pick a conservative value of $\rho = 10$.

7.3 Discussion of the Results

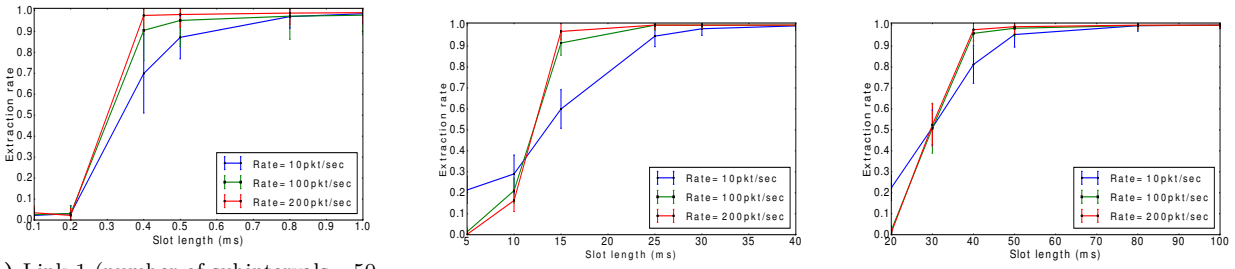
To fingerprint a flow, we randomly choose a flow delay from our delay database, and a flow from the Poisson distribution database. We embed the fingerprint on the flow and then add the delay to the flow. We finally use the extractor (who knows the fingerprinting key) on the noisy flows. We embed various number of bits per interval on various links as they have different noise standard deviations; specifically, n_{bit} is 5, 2, and 1 for links 1, 2 and 3, respectively.

Figure 9 shows the extraction results for the 3 selected links. As expected, increasing slot length increases the extraction rate. Also, higher packet rates result in better extraction with the same parameters, as they offer more packets to be fingerprinted.

Comparing with analysis. Figure 10 compares our simulation results with that of our analysis from Section 6. As can be seen, our analysis presents an upper-bound on the experimental extraction rate. This is due to the imperfection of our modeling of network noise (as discussed earlier), and also the artifacts not included in our model such as bursty errors due to temporary network conditions. Note that our analysis does *not* aim at tightly predicting the performance; instead, it intends to (1) demonstrate how various parameters trade off TagIt 's features (like FP, invisibility, etc.), and (2) enable picking the values of TagIt's parameters to be used in the experiments for various traffic rates and network conditions.

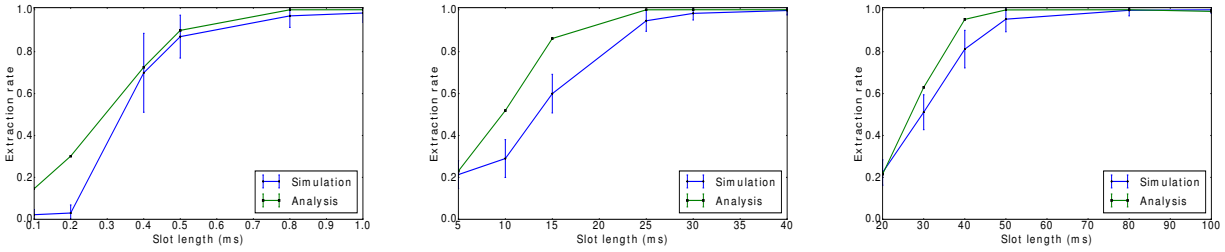
8 Implementation

We implement TagIt on Linux to fingerprint live network flows. Our implementation is done in C++ using NFQUEUE and libnetfilter libraries [25]. We did our experiments on two different links: Link 1 from simulations, and Link 4: Campus-California with standard deviation of delay in the range $9.773 - 12.347$ ms. We



(a) Link 1 (number of subintervals= 50 − 300) **(b)** Link 2 (number of subintervals= 10) **(c)** Link 3 (number of subintervals= 10)

Fig. 9. Extraction result for different nodes in Europe and Asia (no coding)



(a) Link 1 **(b)** Link 2 **(c)** Link 3

Fig. 10. Comparing analysis and simulation results for rate= 10.

use the communication system toolbox from Matlab to implement our coding algorithms. We use two coding algorithms in our experiments: Reed-Solomon codes (RS) and Convolutional codes [26]. See Appendix A for introduction on these coding algorithms.

On each of the links, we try various coding algorithms and rates to achieve an extraction rate close to 1. Note that our choice of coding algorithms and parameters are not optimal, but to demonstrate the possibility of compensating for remaining network errors through coding. We leave the investigation of optimal codes to future work.

In link 1, we embed $n_{bits} = 5$ bits in each fingerprint interval. Therefore, we use a Reed-Solomon code to correct the errors on this link. This is because, as discussed in the appendix, RS codes are best for correcting bursty errors. We aim at fixing 5 bits of errors; we therefore use a rate 2/3 RS code with parameters $n = 31$ and $k = 21$.

On the other hand, in link 4 we embed $n_{bits} = 1$ bit per interval. We therefore use the Convolutional code on this link, which works better on sparse errors (however, it works best on longer streams). Similarly aiming to correct 5 bits of errors, we try two rates of 1/3 and 2/3 with constraint lengths of 3 and [5,4], respectively. We use a Truncated termination mode for our decoder.

Table 4 summarizes the results on live traffic for various parameters and flow rates on the two links. The results are averaged for 100 fingerprinted flows. We pick various parameters for different tests to demonstrate the impact of parameters on the performance.

9 Fingerprint Invisibility

In this section, we evaluate TagIt's invisibility.

9.1 Kolmogorov-Smirnov Similarity Test

The Kolmogorov-Smirnov test compares the empirical distribution of two samples and based on their maximum distance decides if they are from the same distribution. We use the KS test to distinguish between TagIt fingerprinted flows and their non-fingerprinted versions. Table 5 summarizes our experiments for various parameters and flow rates. For each set of parameters (each row of the table) we generate 100 fingerprinted flows and compare each fingerprinted flow with its non-fingerprinted version using the KS test. Table 5 shows

Table 4. Running TagIt on live traffic.

	λ (flow rate)	T (ms)	r	m_ℓ (ms)	R_{move}	r_c (coding rate)	Ave. extraction rate	Ave. extraction after coding
Link 1	$\lambda = 10$	1800	600	0.5	1	2/3	0.946	1
	$\lambda = 100$	270	90				0.951	1
Link 4	$\lambda = 10$	1800	15	20	1	2/3	0.82	0.996
	$\lambda = 10$	1920	16		0.6	1/3	0.943	0.98
	$\lambda = 10$	2160	18		1		0.96	1
	$\lambda = 100$	2160	18		0.6		0.93	1

the result of KS test on different links with their confidence level. The "Pass rate" column shows the fraction of fingerprinted flows (out of 100) that are declared to be from the same distribution as their non-fingerprinted version by the KS test (a higher pass rate means better invisibility). The table also demonstrates the trade-off between invisibility and extraction performance, e.g., for various interval lengths, number of fingerprint bits, and the ratio of the moved packets.

Also, as described earlier, R_{move} (the fraction of packets being delayed) trades off fingerprinting performance with invisibility. Figure 11 shows the impact of R_{move}. As can be seen, increasing R_{move} improves fingerprinting performance (increases extraction rate and reduces false positive) at the price of degrading invisibility (increasing the KS statistic).

9.2 Single flow invisibility

We also use the delay imposed during fingerprinting as another metric to evaluate invisibility. Such an attack can be performed by an adversary who feeds his own flows into the fingerprinting system in order to measure fingerprint perturbations. The worst-case delay per packet inserted by TagIt fingerprinter is given by $max_{delay} = \frac{T}{r}(2 - \frac{2}{m})$. Table 6 shows the average of max_{delay} for different r.

9.3 Multi-flow invisibility

We also evaluate TagIt's invisibility to the MFA attack of Kiyavash et al. [22], discussed earlier. As mentioned before, TagIt uses randomization to defeat MFA. Because of using randomization by TagIt, fingerprinted flows will have different patterns even when the fingerprint key is the same. Figure 12 shows the cumulative histogram for 10 flows in an interval before and after fingerprinting, for different slot lengths. As can be seen, even the use of 10 flows can not leak the fingerprint since TagIt's randomization spreads packets evenly across various slots.

9.4 Fingerprinting key entropy

A trivial way to attack a fingerprinting system is to guess its fingerprinting key. Therefore, the key should have a high-entropy making it practically infeasible to be guessed. When TagIt embeds 1 bit per interval, the space of fingerprinting keys is the set of all possible permutation functions Π^0 and Π^1. Thus, the number of possible permutations is given by:

$$k_{space} = (((m)!)^{rn})^2 \tag{13}$$

Also, we have m different choices for the random seed of each interval, d_k. So the key entropy of our fingerprinting scheme is:

$$log_2 k_{space} m \tag{14}$$

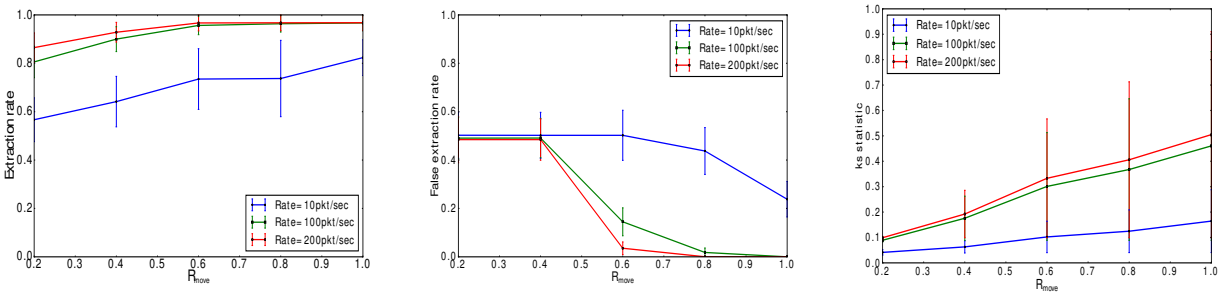
Evaluating this for the parameters of Table 4 results in a key with over 7552 bits of uncertainty, which is significantly costly to be guessed.

10 Future Directions

Comprehensive Invisibility Analysis. While in Section 9 we evaluated TagIt's invisibility against specific classifiers, our analysis does not offer theoretical guarantees on the invisibility of fingerprints against other types of classifiers. Future research may deploy modern classification algorithms (including machine

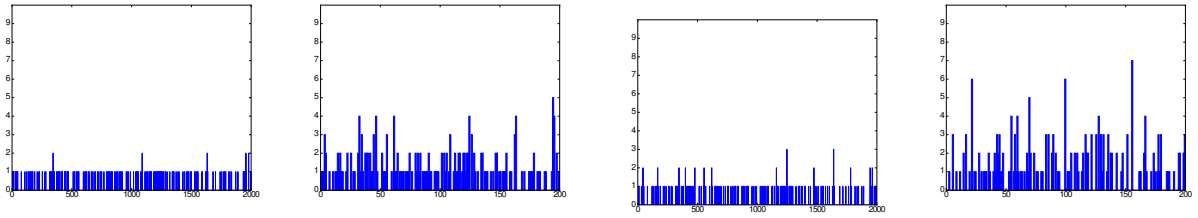
Table 5. Kolmogorov-Smirnov test results for various parameters

Link	λ(flow rate)	T (ms)	r	m_ℓ(ms)	R_{move}	Confidence level	Pass rate			Extraction rate
							$n = 30$	$n = 20$	$n = 10$	
Link 2	$\lambda = 10$	360	10	6	0.6	95%		1		0.52
		480		8			0.99	1	1	0.688
		600		10			0.92	0.98	1	0.745
		360		6	0.8	95%		1		0.56
		480		8			0.94	0.99	1	0.71
		600		10			0.60	0.85	0.98	0.819
		360		6	1	95%	0.87	0.99	1	0.68
		480		8			0.25	0.53	0.91	0.85
		600		10			0.0	0.15	0.64	0.89



(a) Extraction rate **(b)** False Positive **(c)** Invisibility

Fig. 11. The impact of R_{move} on invisibility and extraction performance (slot length= $8ms$).



(a) Non-Fingerprint (slot= 2ms) **(b)** Non-Fingerprint (slot= 20ms) **(c)** Fingerprinted (slot= 2ms) **(d)** Fingerprinted (slot= 20ms)

Fig. 12. Cumulative histogram of 10 flows, non-fingerprinted and fingerprinted with different slot lengths (quantization step in each figure is half of slot length). (a) and (c) have 2000 points in the X axis because slot length is 2 vs. 20 in (b)and (d), and interval length is $2sec$.

Table 6. Average and maximum fingerprint delay per packet for different r parameters

r	T/r (ms)	Average delay (ms)	Maximum delay (ms)
10	180	84.11	300
20	90	40.55	150
30	60	28.15	100
60	30	14.44	50

learning algorithms or statistical tests) that are able to distinguish TagIt fingerprints with higher confidence than the classifiers used in this paper. Therefore, an avenue for future work is to evaluate TagIt's invisibility against other classifiers, and in general seek provable guarantees on the invisibility of fingerprints. This may in turn lead to design adjustments that improve TagIt's invisibility.

Experiments on Tor. Compromising anonymity systems like Tor is one of the potential use cases of flow fingerprints, as we argued early in the paper to motivate the importance of the flow fingerprinting problem. However, our experiments throughout this paper were mainly performed on Planetlab nodes, representing reg-

ular Internet nodes, but not on the Tor network. This is due to two reasons: First, based on our measurements of Tor traffic, the behavior of latency in Tor is significantly different than regular Internet. Therefore, our theoretical analysis of performance (e.g., Section 6) is not valid when fingerprints are used on Tor traffic. In particular, while we modeled the network jitter of regular traffic using Laplacian distribution (which simplifies our derivations), our measurements show that Tor jitter has a unique distribution that highly deviates from that of normal traffic. One avenue for future work is to derive accurate statistical models for traffic behavior in Tor (and similar anonymity systems), and use such a model to analytically evaluate the performance of a fingerprinting system like ours when used in Tor.

Second, our measurements on Tor shows that network jitter varies drastically across different Tor circuits. As shown in our experiments and demonstrated through analysis, the performance and invisibility of TagIt is highly sensitive to the choice of parameters and network conditions. However, the fingerprinting entities are not aware of all of the Tor relays comprising a particular Tor connection, consequently, they can not tailor the fingerprinting parameters to each specific Tor connection. This makes the use of TagIt unsafe in the Tor application as the fingerprinting parties can not adjust the right invisibility-performance balance, e.g., the fingerprint may be overtly visible, or highly unreliable due to the use of improper fingerprinting parameters. As a future research direction, we suggest designing flow fingerprinting systems that are less sensitive to the network conditions, therefore usable on Tor even though the fingerprinting parties are unaware of the relays comprising connections.

11 Conclusion

We designe the first blind flow fingerprinting system called TagIt. We extensively evaluate the performance and invisibility of TagIt through theoretical analysis. We also evaluate TagIt through extensive simulations on network traces as well as through experimenting over live network traffic.

Acknowledgments. We would like to thank anonymous reviewers and our shepherd, George Danezis, for feedback and suggestions. This work was supported by the NSF grant CNS-1525642.

References

[1] R. Archibald and D. Ghosal. A covert timing channel based on fountain codes. In *International Conference on Trust, Security and Privacy in Computing and Communications*, pages 970–977, 2012.

[2] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding, 4th International Workshop, IHW 2001, Pittsburgh, PA, USA, April 25-27, 2001, Proceedings*, pages 245–257, 2001.

[3] B. A. Bash, D. Goeckel, D. Towsley, and S. Guha. Hiding information in noise: Fundamental limits of covert wireless communication. *IEEE Communications Magazine*, 53(12):26–31, 2015.

[4] A. C. Bavier, M. Bowman, B. N. Chun, D. E. Culler, S. Karlin, S. Muir, L. L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating systems support for planetary-scale network services. In *1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California, USA, Proceedings*, pages 253–266, 2004.

[5] A. Blum, D. X. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings*, pages 258–277, 2004.

[6] S. Cabuk. *Network covert channels: Design, analysis, detection, and elimination*. PhD thesis, Purdue University, Jan. 2006.

[7] G. Danezis. The traffic analysis of continuous-time mixes. In *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*, pages 35–50, 2004.

[8] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320, 2004.

[9] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *RAID*, pages 17–35, 2002.

[10] B. P. Dunn, M. Bloch, and J. N. Laneman. Secure bits through queues. In *Networking and Information Theory, 2009. ITW 2009. IEEE Information Theory Workshop on*, pages 37–41. IEEE, 2009.

[11] J. A. Elices and F. Pérez-González. The flow fingerprinting game. In *2013 IEEE International Workshop on Information Forensics and Security, WIFS 2013, Guangzhou, China, November 18-21, 2013*, pages 97–102, 2013.

[12] J. A. Elices and F. Pérez-González. A highly optimized flow-correlation attack. *CoRR*, abs/1310.4577, 2013.

[13] B. F. U. Filho, R. D. Souza, C. Pimentel, and M. Jar. Convolutional codes under a minimal trellis complexity measure. *IEEE Trans. Communications*, 57(1):1–5, 2009.

[14] G. Garrammone. On decoding complexity of reed-solomon codes on the packet erasure channel. *IEEE Communications Letters*, 17(4):773–776, 2013.

[15] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia. Model-Based Covert Timing Channels: Automated Modeling and Evasion. In R. Lippmann, E. Kirda, and A. Trachtenberg, editors, *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*, volume 5230 of *Lecture Notes in Computer Science*, pages 211–230. Springer, 2008.

[16] A. Houmansadr and N. Borisov. CoCo: Coding-Based Covert Timing Channels for Network Flows. In *The 13^{th} Information Hiding Conference (IH)*, 2011.

[17] A. Houmansadr and N. Borisov. SWIRL: A scalable watermark to detect correlated network flows. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.

[18] A. Houmansadr and N. Borisov. BotMosaic: Collaborative network watermark for the detection of IRC-based botnets. *Journal of Systems and Software*, 86(3):707 – 715, 2013.

[19] A. Houmansadr and N. Borisov. The need for flow fingerprints to link correlated network flows. In *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*, pages 205–224, 2013.

[20] A. Houmansadr, N. Kiyavash, and N. Borisov. RAINBOW: A robust and invisible non-blind watermark for network flows. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, 2009.

[21] A. Houmansadr, N. Kiyavash, and N. Borisov. Non-Blind Watermarking of Network Flows. *IEEE/ACM Transactions on Networking*, 22(4):1232–1244, Aug 2014.

[22] N. Kiyavash, A. Houmansadr, and N. Borisov. Multi-flow attacks against network flow watermarking schemes. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 307–320, 2008.

[23] K. Kothari and M. Wright. Mimic: An active covert channel that evades regularity-based detection. *Computer Networks*, 57(3):647–657, 2013.

[24] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright. Timing attacks in low-latency mix systems (extended abstract). In *Financial Cryptography, 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers*, pages 251–265, 2004.

[25] Libnetfilter queue. http://www.netfilter.org/projects/libnetfilter_queue.

[26] J. H. V. Lint. *Introduction to Coding Theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 1998.

[27] Y. Liu, D. Ghosal, F. Armknecht, A.-R. Sadeghi, S. Schulz, and S. Katzenbeisser. Hide and Seek in Time - Robust Covert Timing Channels. In M. Backes and P. Ning, editors, *European Symposium on Research in Computer Security (ESORICS)*, volume 5789 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2009.

[28] Y. Liu, D. Ghosal, F. Armknecht, A.-R. Sadeghi, S. Schulz, and S. Katzenbeisser. Robust and Undetectable Steganographic Timing Channels for i.i.d. Traffic. In R. Böhme, P. W. L. Fong, and R. Safavi-Naini, editors, *Information Hiding*, volume 6387 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 2010.

[29] R. J. McEliece and W. Lin. The trellis complexity of convolutional codes. *IEEE Trans. Information Theory*, 42(6):1855–1864, 1996.

[30] P. Peng, P. Ning, and D. S. Reeves. On the secrecy of timing-based active watermarking trace-back techniques. In *2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA*, pages 334–349, 2006.

[31] Y. J. Pyun, Y. H. Park, X. Wang, D. S. Reeves, and P. Ning. Tracing traffic through intermediate hosts that repacketize flows. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA*, pages 634–642, 2007.

[32] D. Ramsbrock, X. Wang, and X. Jiang. A first step towards live botmaster traceback. In *The International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 59–77, 2008.

[33] J.-F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29. Springer-Verlag, LNCS 2009, July 2000.

[34] S. H. Sellke, C.-C. Wang, S. Bagchi, and N. B. Shroff. TCP/IP Timing Channels: Theory to Implementation. In *INFOCOM*, pages 2204–2212, 2009.

[35] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, Berkeley, CA, USA, 2006. USENIX Association.

[36] R. Soltani, D. Goeckel, D. Towsley, , and A. Houmansadr. Covert Communications on Poisson Packet Channels. In *The 53^{rd} Annual Allerton Conference on Communication, Control, and Computing*, 2015.

[37] R. Soltani, D. Goeckel, D. Towsley, , and A. Houmansadr. Covert Communications on Renewal Packet Channels. In *The 54^{th} Annual Allerton Conference on Communication, Control, and Computing*, 2016.

[38] S. Staniford-Chen and T. L. Heberlein. Holding intruders accountable on the internet. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 39–49, 1995.

[39] R. J. Walls, K. Kothari, and M. Wright. Liquid: A detection-resistant covert timing channel based on ipd shaping. *Computer networks*, 55(6):1217–1228, 2011.

[40] X. Wang, S. Chen, and S. Jajodia. Tracking anonymous peer-to-peer voip calls on the internet. In *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 81–91, 2005.

[41] X. Wang, S. Chen, and S. Jajodia. Network flow watermarking attack on low-latency anonymous communication systems. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 116–130, 2007.

[42] X. Wang and D. S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of

interpacket delays. In *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, pages 20–29, 2003.

[43] X. Wang, D. S. Reeves, and S. F. Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *Computer Security - ESORICS 2002, 7th European Symposium on Research in Computer Security, Zurich, Switzerland, October 14-16, 2002, Proceedings*, pages 244–263, 2002.

[44] K. Yoda and H. Etoh. Finding a connection chain for tracing intruders. In *Computer Security - ESORICS 2000, 6th European Symposium on Research in Computer Security, Toulouse, France, October 4-6, 2000, Proceedings*, pages 191–205, 2000.

[45] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao. Dsss-based flow marking technique for invisible traceback. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 18–32, 2007.

[46] Y. Zhang and V. Paxson. Detecting stepping stones. In *9th USENIX Security Symposium, Denver, Colorado, USA, August 14-17, 2000*, 2000.

A Coding Algorithms Used

A.1 Convolutional codes

Convolutional codes are a class of linear codes that have been used in different applications [26]. A Convolutional code is represented by (n, k, L), in which k is the length of the input stream entering the encoder, and n is the length of the output. k/n is the rate of the code. The output is generated by convolving multiple bits of the input by a generator function G. The length of the generator function is the *constraint length*, L. Increasing the constraint length makes the decoding more complex, bur also more powerful. The Viterbi algorithm is an efficient way to decode Convolutional codes [26].

A.2 Reed-Solomon codes

Reed-Solomon codes are a class of linear block codes that are used in digital communications where the noise is bursty. A Reed-Solomon encoder takes a block of data and adds extra "redundant" bits. The Reed-Solomon code is represented by the triple (n, k, m_c), where k is the number of input blocks, $m_c = \log_2(n+1)$ is the length of each block, and n is the number of blocks in the output (the message contains $m_c n$ bits). By adding $2t$ redundant blocks, the RS code is able to correct up to t corrupted blocks. The number and type of errors that can be corrected depend on the characteristics of the Reed-Solomon code. The number of distinct fingerprints that can be embedded and extracted reliably using RS code by TagIt is given by:

$$N = 2^{m_c k} \tag{15}$$

For instance, for $k = 5$, and $m_c = 5$, we have that $N = 10^7$.