1-1-2011

Steganographic Timing Channels

Adam Aviv
University of Pennsylvania, aviv@seas.upenn.edu

Guarav Shah
University of Pennsylvania, gauravsh@cis.upenn.edu

Matt Blaze
University of Pennsylvania, blaze@cis.upenn.edu

Steganographic Timing Channels

Abstract

This paper describes steganographic timing channels that use cryptographic primitives to hide the presence of covert channels in the timing of network traffic. We have identified two key properties for steganographic timing channels: (1) the parameters of the scheme should be cryptographically keyed, and (2) the distribution of input timings should be indistinguishable from output timings. These properties are necessary (although we make no claim they are sufficient) for the undetectability of a steganographic timing channel. Without them, the contents of the channel can be read and observed by unauthorized persons, and the presence of the channel is trivially exposed by noticing large changes in timing distributions – a previously proposed methodology for covert channel detection. Our steganographic timing scheme meets the secrecy requirement by employing cryptographic keys, and we achieve a restricted form of input/output distribution parity. Under certain distributions, our schemes conforms to a uniformness property; input timings that are uniformly distributed modulo a timing window are indistinguishable from output timings, measured under the same modulo. We also demonstrate that our scheme is practical under real network conditions, and finally present an empirical study of its covertness using the firstorder entropy metric, as suggested by Gianvecchio and Wang [8], which is currently the best published practical detection heuristic for timing channels.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-11-18.

Steganographic Timing Channels

Adam J. Aviv, Gaurav Shah and Matt Blaze
University of Pennsylvania
{aviv, gauravsh, blaze}@cis.upenn.edu

Abstract

This paper describes *steganographic timing channels* that use cryptographic primitives to hide the presence of covert channels in the timing of network traffic. We have identified two key properties for steganographic timing channels: (1) the parameters of the scheme should be cryptographically keyed, and (2) the distribution of input timings should be indistinguishable from output timings. These properties are necessary (although we make no claim they are sufficient) for the undetectability of a steganographic timing channel. Without them, the contents of the channel can be read and observed by unauthorized persons, and the presence of the channel is trivially exposed by noticing large changes in timing distributions – a previously proposed methodology for covert channel detection. Our steganographic timing scheme meets the secrecy requirement by employing cryptographic keys, and we achieve a restricted form of input/output distribution parity. Under certain distributions, our schemes conforms to a *uniformness property*; input timings that are uniformly distributed modulo a timing window are indistinguishable from output timings, measured under the same modulo. We also demonstrate that our scheme is practical under real network conditions, and finally present an empirical study of its covertness using the first-order entropy metric, as suggested by Gianvecchio and Wang [8], which is currently the best published practical detection heuristic for timing channels.

1 Introduction

A network timing channel is a useful mechanism for covertly hiding information within network traffic. Previously described timing channel encoding schemes make no explicit attempt to hide their presence within network traffic. As a consequence, a network monitor can detect timing channels by observing the timing of network events.

In this paper, we describe a *steganographic timing channel* that makes use of cryptographic primitives to covertly hide the presence of the channel within the inter-packet timings of legitimate network traffic. The goal of a steganographic timing channel is to use an encoding such that the covertness and security of the channel does not depend on the secrecy of the encoding algorithm, but rather on a shared secret between the covert channel sender and the receiver (eavesdropper). Note that this closely parallels a fundamental principle of cryptography - the secrecy of the ciphertext should not depend on the secrecy of the algorithm, but rather on its input parameter, the cryptographic key [11].

Two key properties are required to meet these goals: (1) cryptographically keyed parameters and (2) the distribution of output timings should be indistinguishable from the distribution of the input timings. These are necessary properties (although we make no claims that they are sufficient) for the undetectability of a covert stenographic channel. The first property ensures not only that the channel contents not be read by an unauthorized entity but also that its presence cannot be detected without knowledge of the secret. The second property ensures that the distribution of input events (be it bits, or timings) well matches the distribution of the encoded output. Distribution change detection schemes have been shown to be an effective means of detecting covert and steganographic channels [3, 4, 8].

Our steganographic channel achieves secrecy protection by employing a shared cryptographic key to produce a pseudo-random sequence. Without knowledge of the key, a network observer cannot effectively detect the channel or decode its contents. We also achieve a restricted form of input/output distribution parity. Under certain distributions, our schemes conforms to a *covert uniformness property*; input timings that are uniformly distributed modulo a timing window are indistinguishable from output timings, measured under the same modulo. Our encoding is resistant to detection schemes that rely on identifying a change from a uniform to a more regular timing distribution, and we provide analytic reasoning to this effect.

Our encoding relies on modifying inter-packet delays in existing network traffic to embed information. The modulation scheme is built upon the JitterBug encoding – proposed but not analyzed – by Shah et al. [17]. In contrast with classic steganographic schemes, our channel encoder does not control message events but rather modulates exiting events. As such, we address many practical concerns, namely the synchronization of the sender and receiver in the presence of network events. We evaluate the scheme under real network conditions using collected keyboard timings for ssh, and our evaluation demonstrates that the steganographic covert channel provides robust decoding in the presence of packet loss and network jitter.

The rest of this paper is organized as follows. Section 2 describes related work. We describe the original JitterBug encoding scheme in Section 3. The steganographic encoding scheme and its covertness properties are described in Section 4. We address the practical issues of channel decoding under the presence of network effects and errors in Section 5. We next describe the entropy detection scheme followed by a discussion of the problems with its model and previous evaluation in Section 6. Our experimental evaluation is presented in Section 7 and we conclude in Section 8.

2 Related Work

Timing Channels. Various encoding mechanisms have been proposed for embedding covert timing channels in network traffic. Cabuk et al. [5] describe the design of a network timing channel in IP network traffic. The *simple* IP covert timing channel uses a packet-counting encoding where the reception or absence of a network packet during a fixed timing interval signifies a bit of information. Cabuk also describes a *time replay* covert timing channel [4] where a set of previously recorded inter-arrival times are replayed and the covert receiver maps these to symbols of information. The inter-packet delay to symbol mapping is assumed to be shared in advance between the covert sender (performing the replay) and receiver.

Luo *et al.* [12] describe the TCPScript timing channel that encodes information using a burst of consecutive back-to-back TCP segments. Manipulation of inter-packet delays has also been used to add watermarks to regular network traffic. Wang *et al.* [22, 23] describe a watermark embedding based on slightly modifying the inter-packet delays of a group of network packets. A decoder can then extract the watermark by observing the changes in statistical properties of packet timings.

In previous work, the JitterBug covert timing channel [17] was described. This channel embeds bits of information by slightly modifying the inter-arrival times of existing network traffic. The inter-packet delays are modified such that they satisfy certain properties that a decoder can then use to extract the encoded information. A modification to the encoding scheme was also presented that can hide the presence of patterns when using visual inspection of inter-arrival times. However, this scheme alone does not prevent detection against more sophisticated detection algorithms such as entropy (described below).

Standard countermeasures for timing channels are well explored in the literature [25] and are no different when attempting disrupt the steganographic timing channel. However, any countermeasure that attempts to limit the bandwidth of the channel also affects the benign cover traffic, poising a well understood trade-off. Of course, any targeted countermeasure, would need to first suspect a covert timing channel is being used, which, as we will demonstrate, can be very difficult.

Detection. Covert channel detection is a difficult problem in general. Detection algorithms rely on the notion of *regularity* that is a property of the specific channel encoding. Cabuk *et al.* [5] propose using the ϵ-similarity and the variance in the network inter-packet delays to detect a covert channel. Their method, however, only works for the simple IP network timing channel.

Berk *et al.* [3] propose a method for detecting channel encodings in network packet delays assuming the channel is used at ideal capacity. The distribution of inter-packet delays that achieve this optimal capacity is computed based on the current network conditions. Now, if the observed inter-packet delays satisfy this distribution, the traffic is flagged as covert. Again, this technique only works for the specific encoding described in this work.

Gianvecchio and Wang propose using corrected conditional entropy and first-order entropy of inter-arrival times to detect various kinds of network timing channels [8]. Interestingly, this is the only approach that proposes the use of a quantitative heuristic for detecting covert channels in general. However, a problem with this specific approach is that it requires advanced training using a potentially large sample of legitimate network traffic. Moreover, these

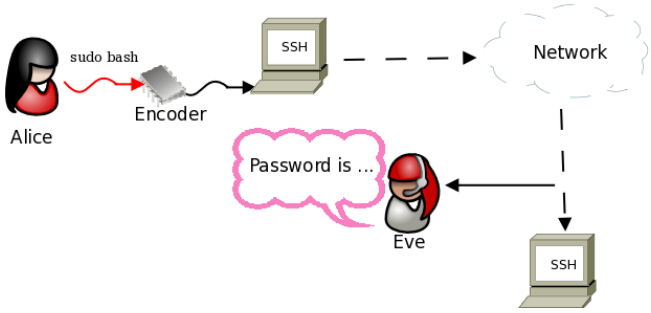

Figure 1: Steganographic encoding setup.

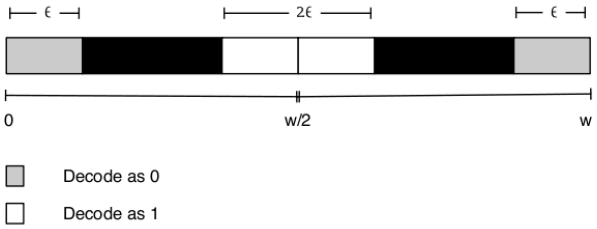

Figure 2: Timing Window with Error Bounding (ϵ) used at decoding for basic JitterBug (Source: [17]).

parameters may vary depending on the characteristics of the network traffic, the application, and the location of the detector within the network.

Smith and Knight [19] present a measure for the probability of covert channel detection based on the *exploit symbol to noise ratio* (SNR). This metric, although novel, is only applicable to covert storage channels. Storage channels for network protocols attempt to hide information within the packet header fields [10, 14, 16]. Smith and Knight's SNR metric observes that a storage channel must use a set of exploit symbols, a subset of the possible values for the chosen protocol field, to represent the covert data. The exploit symbols are then artificially injected into legitimate network traffic to disseminate a message. A covertness metric can be derived from the exploit SNR which is defined as the ratio between the proportion of injected exploit symbols to the proportion at which they occur naturally in legitimate traffic. A low SNR, therefore, implies high covertness at the expense of reliability, while a high SNR implies low covertness with a higher transmission rate and reliability. However, in contrast with a storage channel where there are only a finite set of discrete values for a given protocol field available for encoding, the steganographic timing channel may use an infinite number of continuous packet inter-arrival times for performing the encoding. Therefore, Smith and Knight's SNR metric is not useful for analyzing the covertness of the steganographic timing channel.

Formal Models of Steganography. One of the earliest model of steganography is described by Simmons as the "Prisoner Problem" in 1983 [18]. Alice and Bob are prisoners locked up in different cells and wish to plan an escape. They are allowed to communicate via a courier as long as their messages are innocuous, but the warden is privy to all messages and wants to incriminate the two prisoners further and thwart any escape. Alice and Bob, sharing a secret prior to incarceration, succeed if they can plan their escape without rousing the suspicion of the warden. To solve this problem, Simmons introduced the notion of a *subliminal channels*; a means of communication that is in full view of an adversary, yet is unnoticeable. From this model, steganography has been formalized based on the passive eavesdropper (i.e., the warden) in terms of complexity theory [9], as a public key system [21, 2] and using information-theoretic models [13, 26, 6].

Most closely related to our approach, information-theoretic formulations are interesting because secrecy is modeled based on the observable changes in the probability distributions of symbols between encoded and un-encoded messages. For example, Cachin [6] defines a stegosystem to be ϵ-secure against a passive adversary if the difference in probability distribution between the cover text and the steganographic text is bounded by ϵ. A perfectly secure stegosystem ($\epsilon = 0$) can be constructed under this model, except that it assumes that the cover text is sampled from a perfectly uniform distribution, which is unreasonable in practice. As we will discuss in Section 4.1, we assume a restricted form, that the cover text is uniformly distributed modulo a window, and our encoder produces output that meets the same requirement.

In general, these models either assume that the input symbol probability distribution is known *a priori*, or that the steganographic encoder can arbitrarily generate messages or modify symbols in messages to match the input distributions. In contrast, our steganographic timing channel can only modulate pre-existing events rather then generating new events, and thus these models are less applicable due to the semi-passive nature of the timing channel.

3 JitterBug Encoding

The JitterBug timing channel embeds a timing channel within network traffic by manipulating inter-packet delays of *existing* network packets. In previous work, the Keyboard JitterBug [17] was described as a hardware keystroke logger placed on the input line between the keyboard and the computer. JitterBug adds minute delays to keystroke presses to create a timing channel while the user is typing in an interactive network application such as SSH. These modified keystroke timings get propagated in the form of encoded inter-packet network delays, creating a network timing channel. The covert receiver monitors inter-arrival times somewhere along the network path and can decode and extract information from the channel such as typed passwords sent by the JitterBug. A diagram of the JitterBug setup is presented in Figure 1.

3.1 Basic Encoding Scheme

In the basic encoding scheme, a sequence of bits $\{b_i\}$ is embedded into network packet timings $\{t_i\}$ by introducing a delay τ_i for each packet such that the resulting timings $\{t'_i\}$ encode a bit of information within the modified inter-packet delays $\delta_i = t'_i - t'_{i-1}$. At decoding:

$$\delta_i \bmod w = \begin{cases} 0 \pm \epsilon \pmod w & \text{if } b_i = 0 \\ \lfloor \frac{w}{2} \rfloor \pm \epsilon \pmod w & \text{if } b_i = 1 \end{cases}$$

where w is the *timing window* and ϵ is the decoding error bound. So the delays can be calculated:

$$\text{if } b_i = 0 : \tau_i = w - \hat{\delta}_i \pmod w \qquad \text{if } b_i = 1 : \tau_i = \begin{cases} \frac{w}{2} - \hat{\delta}_i \pmod w & \text{if } \hat{\delta}_i \pmod w < \frac{w}{2} \\ \frac{3w}{2} - \hat{\delta}_i \pmod w & \text{else} \end{cases}$$

where $\hat{\delta} i$ is the unmodified inter-packet delay.

For example, to encode a bit 0, the inter-arrival time is quantized to a multiple of the window size w. On the other hand, the encoding of a bit 1 is performed by adding a delay such that the inter-arrival time is $\frac{w}{2}$ modulo the window size. Additionally, in order to handle network jitter, the basic scheme employs an error bound of ϵ, so if the observed δ_i is within the bound, it can still be decoded. Figure 2 (Source: [17]) shows the timing window used for decoding with ϵ bounding to handle network jitter.

The above timing channel can be detected by visual inspection of network inter-arrival times since encoding causes clustering around 0 and $\frac{w}{2}$ (modulo w). To overcome this shortcoming, an extension was proposed to the basic encoding scheme. Instead of an encoding that relies on changing inter-packet delays modulo the window size to fixed values, i.e., 0 and $\frac{w}{2}$, there can be multiple values that can represent 0 and 1. However, there is no discussion of any practical encoding scheme based on this idea and, in particular, does not address how the covert sender and receiver can synchronize with one another.

3.2 Rotating Window Encoding Scheme

As before, $\{b_i\}$ denotes the binary sequence to be transmitted and $\{t_i\}$ is the sequence of input network packet times. $\{s_i\}$ is a pseudo-random sequence of integers that range from 0 to the window size w. The sequence $\{s_i\}$ is only known to the covert sender and receiver and acts as a shared secret.

Like before, the window is divided into two equal parts, but instead of splits being at 0 and $\frac{w}{2}$, the window is split at s_i and its antipode, $s_i + \frac{w}{2} \pmod w$. With this encoding, time delays will not consistently cluster around 0 and its antipode, $\frac{w}{2}$, but rather around $\{s_i\}$ and its associated antipodal sequence. More precisely, to transmit a bit b_i, the delay added is such that at decoding:

$$(\delta_i - s_i) \bmod w = \begin{cases} 0 \pm \epsilon \pmod w & \text{if } b_i = 0 \\ \lfloor \frac{w}{2} \rfloor \pm \epsilon \pmod w & \text{if } b_i = 1 \end{cases}$$

where $\delta_i = t'_i - t'_{i-1}$ is the modified inter-arrival times of network packets. Like before, this scheme employs error bounding during decoding using ϵ to deal with network jitter.

Although the addition of a rotating windows has an effect on the visual representation of observed inter-arrival times, i.e., they no longer cluster around multiples of $\frac{w}{2}$, the rotating window scheme as described previously does not provide covertness against more sophisticated detection schemes. Moreover, the practical issues of encoding and decoding in the presence of noise such as network jitter and packet losses are not addressed. It is also unseen if the

previous encoding meets the covert steganographic properties describe previously, in particular the low input/output distribution parity. In the next section, we present a drop-in replacement for this scheme that is practical in real network conditions and resistant to timing channel detection algorithms. Additionally we will discuss the properties of our new encoding scheme.

4 Steganographic Encoding

The simple rotating window encoding scheme rotates the encoding window by a pseudo-random integer known only to the covert sender and receiver for every subsequent symbol that is transmitted. In the original scheme, these rotations are chosen in the range of 0 to $w-1$. In our steganographic scheme, we allow rotations to have an even finer granularity, and their range is chosen irrespective of the window size. The granularity of rotations is denoted by l, referred to as the number of *rotation splits*. l defines the number of divisions or splits by which an encoding window can be rotated. For example, with $l = 100$, the encoding window has 100 possible configurations for each symbol transmission. In addition, the shared pseudo-random sequence of rotations are generated in a cryptographically secure manner using a shared key between the covert sender and receiver.

Let S be the cryptographically keyed pseudo-random sequence of rotations. Each $s_i \in S$ is chosen in the range $\{1 \dots l\}$ and describes a rotation as a fraction of the window w. The encoding of a bit 0 takes place at delays of $w * (\frac{s_i}{l})$ (modulo w) and the encoding of a bit 1 takes place at the antipode, $w * (\frac{s_i}{l}) + \frac{w}{2}$ (modulo w). For simplicity of notation, instead of describing s_i as a whole number fraction $\frac{s_i}{l}$, we henceforth assume that s_i is already in its fractional form. Now, the encoding and decoding routine is exactly as described in Section 3.2, with rotations being performed at a much finer granularity. Additionally, decoding can still make use of the ϵ error bounding to deal with network jitter.

With splits alone, the encoded output (modulo w) will still experience clustering, but instead of clustering around multiples of $\frac{w}{2}$, now the clustering occurs at multiples of $\frac{w}{l}$. To alleviate this problem, we introduce a uniformly random value c (s.t. $-\frac{w}{2}l \leq c < \frac{w}{2}l$) that is added to encoded output inter-arrival times. Therefore, the encoding is performed such that at decoding:

$$(\delta_i + s_i) \bmod w = \begin{cases} 0 \pm \frac{w}{2l} + \epsilon \pmod w & \text{if } b_i = 0 \\ \lfloor \frac{w}{2} \rfloor \pm \frac{w}{2l} + \epsilon \pmod w & \text{if } b_i = 1 \end{cases}$$

And the delays can be calculated as follows:

$$\text{if } b_i = 0: \quad \tau_i = w - (\hat{\delta}_i + s_i) + c \pmod w$$
$$\text{if } b_i = 1: \quad \tau_i = \begin{cases} \frac{w}{2} - (\hat{\delta}_i + s_i) \pmod w + c & \text{if } \hat{\delta}_i + s_i \pmod w < \frac{w}{2} \\ \frac{3w}{2} - (\hat{\delta}_i + s_i) \pmod w + c & \text{else} \end{cases}$$

Where, again, $\hat{\delta}_i$ is the unmodified inter-packet delay.

Now, every encoded inter-arrival time (modulo the timing window) will be uniformly distributed within the window. The resulting variance is negligible compared to network effects and does not hinder decoding using the error bound ϵ or synchronization techniques described in Section 5. Naturally, when $s_i = w$, the steganographic encoding reduces to the rotating window scheme, but as we show, any change in rotation granularity has a considerable impact on the covertness of the channel.

4.1 Covert Uniformness Property

There are no universally applicable metrics for evaluating the covertness of a timing channel encoding. This has generally been based on the obscurity of the technique rather than the algorithmic properties (e.g., the use of a secret key). Watermarking schemes [15] have established some metrics, but the focus is on preserving the watermark content rather than masking its presence. Without such standards, we evaluated the covertness empirically using the only quantitative heuristic available to us in the literature, namely entropy.

In the literature, two primary approaches have been used to theoretical define the secrecy of a steganographic channel. Hopper *et al.* used complexity theory in [9], and defined the insecurity of a channel in terms of the number of steps and queries to an oracle that must be made to detect the encoding. However, we apply a more information theoretic approach, akin to [6], where the secrecy of the channel is based on unobservable changes in input/output distribution.

In Cachin's model, the encoder can manipulate the symbols within the covertext such that the encoding does not change the distribution. This is not possible given our channel's setup because the only action possible is the modulation of existing timing events. Moreover, there is no way of knowing the input distribution given the encoder cannot choose the covertext. As such, we can provide a restricted form of input/output distribution parity. Under certain distributions, our scheme conforms to a *covert uniformness property*; input timings that are uniformly distributed modulo a timing window (w) are indistinguishable from output timings measured under the same window. We acknowledge the limitations of this measure.

Observe that this is a theoretically weaker property then that described by Cachin which considers the distribution in total, and we make no claims that covert uniformness is a sufficient property for undetectability against any arbitrary adversary. However, it is at least sufficient against an adversary performing analysis at a granularity modulo a threshold, and we also argue that this is sufficient in practice and effective against the best known detection scheme available, namely entropy.

The analysis proceeds by using a model similar to that of Simmons'[18]. A warden can observe the network timings and wishes to determine if a steganographic timing channel is present, and if so, determine the contents of the channel. Moreover, the warden can compare timing events to a previously measured legitimate distribution, one that is known not to contain a steganographic timing channel, and can compare the observed distribution at varied granularity.

At the highest granularity, greater than w, the warden could see a shift in distribution mean of at most w, since that is the maximal amount of delay. Note that delays can be added in the range from $[0, w]$, independent of l and c, and thus the expected delay is $\frac{w}{2}$. If the variance of normal network events is greater than $\frac{w}{2}$, the shift would be indistinguishable from normal network events[1].

At a granularity less than the window size, our stenographic encoding produces output that is uniformly distributed modulo w. Observe first that delays are added such that clustering occurs (modulo w) at multiples of $\frac{w}{l}$. Secondly with the addition of c, the resulting inter-arrival times become uniform. In other words, the split distributions provide a discrete uniform distribution within the timing window, and the addition of c makes it continuous. If the original inter-arrival times are uniformly distributed modulo the window size, then the output inter-arrival times are indistinguishable from the input at the granularity modulo the window size.

Since our encoding is a modulation scheme and only affects preexisting events by a maximal amount, the highest order bits of the distribution remain mainly unaffected and are upper bounded by w. In the lower order bits, our scheme provides a stronger condition, namely covert uniformness. A timing sequences whose inter-arrival times are uniformly distributed modulo w will retain this property after encoding. Thus, if the warden is unable to predict the output of the cryptographic pseudo-random sequence, our steganographic encoding is indistinguishable from the legitimate distribution at high and low granularity.

4.2 Channel Secrecy Property

The bounds on the change in distribution discussed in the previous section provided protection against detection. In this section we will discuss channel secrecy in the face of an attacker with knowledge of w and l. Given a sequence of inter-arrival times in which the steganographic encoding exists and with knowledge of w and l, the warden can attempt to infer the rotation sequence and decode the sequence. However, each inference can assume two possible values, and thus every encoding can take two possible values, 0 and 1. Without knowledge of the cryptographically keyed pseudo-random sequence, even with knowledge of w and l, a warden is unable to properly decode the steganographic channel.

Note that a given rotation value s_i corresponds to two equal length encoding locations within the window, $w * s_i$ (modulo w) and its antipode $w * s_i + \frac{w}{2}$ (modulo w)[2]. It is then sufficient to show that for a given encoded inter-arrival either such encoding locations are equally likely, i.e., it could encode either a 0 or a 1 with equal probability, without knowledge of the pseudo-random sequence.

Note that choice of l is critical; l must be even. Otherwise, if $w * s_i + \frac{w}{2}$ (modulo w) might corresponds to a value that cannot possible encode a 0, then the encoding is trivially broken because this inter-arrival obviously would encode a 1.

[1] In our empirical analysis (see Section 7), there seems to be sufficient variance, at least across human keystroke events and HTTP GET requests, such that accurately classifying shifts using entropy detection is very difficult.

[2] Note that we use s_i in its fractional form here

The intuition behind this is simple. The possible encoding locations within the window are $\forall i \in [1 \ldots l],\ i * \frac{w}{l}$ and for a secure choice of l it should be the case that encoding on one side of $\frac{w}{2}$ mirrors those on the other. This is clear, because under a modulo, a valid encoding with the addition of $\frac{w}{2}$ should still be a valid encoding, i.e, $\forall i \in [1 \ldots \frac{l}{2}],\ i * \frac{w}{l} = (i + \frac{l}{2}) * \frac{w}{l}$ where $(i + \frac{l}{2}) \in [1 \ldots l]$. If l is odd, then $(i + \frac{l}{2}) \notin [1 \ldots l]$ and thus a perfect mirror cannot be achieved. Further, if l is even, then clearly $\frac{w}{2}$ must be a possible encoding because $\frac{l}{2} * \frac{w}{l} = \frac{w}{2}$.

Thus, a warden with knowledge of w and l is faced with two equally likely rotations for each bit encoding—one for 0 and one for 1—and cannot distinguish between them without knowledge of the cryptographically derived pseudo-random sequence[3]. This property does not just provide message secrecy, it is also critical for protecting the channel from detection, as discussed previously.

4.3 Choosing a Window and Timing Distribution

We have addressed the two key properties for the steganographic timing channel previously; however, much is dependent on the distribution (modulo w) of the input inter-arrival times. A natural question then is what types of streams can be expected to have this property, and for what values of w? A detailed analysis of various timing distributions that do and do not have this property is beyond the scope of this paper. However, timing distributions based on human-generated input (such as a keyboard) and, more generally, those based on independent events are good candidates.

Clearly, other kinds of input streams will not fall into this category. Machine generated clock pulses, for example, may lack enough low order variations, except perhaps within very small values of w. Keyboard events, however, have very different properties. They have, at their roots, biologically driven events, namely key presses cause by human intention. The essential property is that the *low order* bits of the unmodulated inter-arrival times be uniform.

Polling and quantization effects in the digital domain can also disrupt the uniform distribution property for analog human event timings. For example, if the keyboard driver polls the keyboard buffer for input events at 5ms intervals, the distribution (measured after polling) of inter-keystroke times modulo a window of 20ms will show spikes at multiples of 5ms. Therefore, the covert encoder should be placed such that it can directly modulate *before* these events (and hence, their timings) are polled and processed in the digital domain. This ensures that uniformly distributed output produced by the covert encoder goes through the same distribution changes that affect analog input timings. In the context of the keyboard, the JitterBug must perform the modulation on raw keystroke timings prior to any operating system polling.

Note that even for independent events, the choice of w is still critical here; it must be a small fraction of the normal event inter-arrival times. If w is too large, the inter-arrival times, even of input streams based on independent events (like keyboard input), become non-uniform (modulo w). A choice of w, on the order of tens of milliseconds, seems to produce uniform distributions for the input streams like those generated from the keyboard. This does not imply that choices of w in this range will work for any arbitrary input stream. However, the value of w represents a trade-off. Too large a value risks violating the uniformness property of the input inter-arrival times, and too small a value may lead to increased error rates during decoding. Acceptable values for w therefore depend on the characteristics of the original input stream. Note that we do not propose here a method for deriving the maximum acceptable value of w for any particular stream, only the requirements that w must meet. In our experiments, we used the same value for w as in the original JitterBug, which was found there to be below the limits of human perception but sufficiently high to allow good performance over a network.

5 Steganographic Decoding

The use of a shared pseudo-random sequence complicates the decoding over a network in the presence of noise and errors. There are two broad issues that need to be addressed. First, synchronization of the cipher streams is essential for both the sender and receiver. There is no direct feedback mechanism from the receiver to the sender that can be used for resynchronization, so this needs to be performed by other means. Particularly problematic is the possibility of isolated and bursty deletion errors at the covert decoder due to buffering or packet losses. Insertion errors, falsely identifying a packet as part of the covert channel stream, can also affect synchronization and must be handled as well. Second, the mechanism for resynchronization must also handle general network jitter that can affect the observed

[3]This is analogous to the standard mixing function used in conventional cryptographic stream-cipher encryption where the plain text is XOR'ed with generated bit-stream.

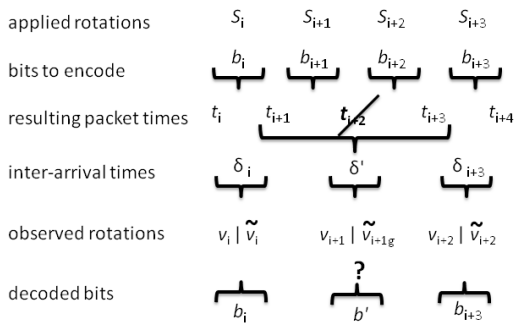

Figure 3: An example of how a deletion error at time t_{i+2} affects the inter-arrival time, δ', and the inferred rotation, $v_{i+1}|\widetilde{v_{i+1}}$ when performing synchronization.

inter-arrival times at the receiver. This requires the decoder to distinguish resynchronization errors (caused due to insertions/deletions) from those caused due to network jitter. Note that during decoding, error bounding can handle network jitter if the pseudo-random streams are in sync at the receiver and sender. However, this does not directly assist synchronization in the presence of errors due to insertions and deletions.

Jitter and Synchronization Errors. Ideally, without network effects or errors, the receiver is synchronized in its shared rotation sequence with the sender, therefore allowing for accurate decoding. In this ideal case, synchronization is achieved by maintaining a count at the receiver of seen inter-arrival times. The count is incremented for every observed packet inter-arrival and is then used as a pointer into the keyed pseudo-random stream, thus ensuring synchronization. Of course, initial synchronization must also be accounted for, but this is easily handled by observing a large window of inter-arrival times to find the starting point in the rotation sequence and can be done in an offline manner. They can also be treated as deletion errors and handled via the mechanisms described below.

Inferring Rotations. The receiver can infer the pseudo-random sequence of rotations V, the inferred rotations, and compare it to S, the set of true rotations. This is done by inspection of the observed inter-arrival times, δ_i, and is performed as follows:

$$v_i = w - \delta_i (\text{mod } w) \text{ if } b_i = 0 \qquad \widetilde{v_i} = \frac{w}{2} - \delta_i (\text{mod } w) \text{ if } b_i = 1$$

The continuous random variable c used during encoding may skew the inferred rotations slightly, but this is handled by inexact matching (described in Section 5) since its effect is negligible compared to network jitter.

The elements of the pseudo-random sequence can be determined based on the knowledge of b_i. Since b_i is unknown, the receiver makes two guesses about the potential values of the applied rotation: v_i, if 0 was encoded and its antipode $\widetilde{v_i}$, if 1 was encoded.

For a sequence of observed inter-arrival times, the receiver generates the set $V = \{(v_i|\widetilde{v_i})\}$ of inferred rotations. From the perspective of the sender, the true rotations form the set $S = \{s_i\}$. With knowledge of the true rotations, the receiver can now compare the set V with the set S to determine if an error occurred. Comparison must be dual matching (denoted by $\approx$) because b_i could be either 0 or 1, i.e., $v_i = s_i$ or $\widetilde{v_i} = s_i$. Note that, if there are no deletion errors and no network jitter (perfect transmission), then $\forall i, (v_i|\widetilde{v_i}) \approx s_i$.

Detecting Deletion and Insertion Errors. When a single deletion error occurs due to packet loss or buffering (e.g., two keystroke packets in an SSH session being sent in the same network packet), the inferred rotation sequence will have two elements missing (compared to what the receiver is expecting). If there was an insertion error, the same property holds in reverse: two elements will be missing from the true sequence when compared to the observed sequence. The discussion below will focus on deletion errors, but all the properties also hold true for insertion errors except that V and S are swapped. Thus, insertion errors can be handled in a similar way. Note that we are ignoring network effects here.

Figure 3 illustrates the properties of desynchronization due to a deletion error. The covert channel sender, encodes bits b_i to b_{i+3}. However, a deletion error occurs and the packet sent at time t_{i+2} and the encoded delay is lost. At this point, the inferred rotation sequence, V, goes out of sync with the true rotation sequence, S.

Since $v_{i+1}|\widetilde{v_{i+1}}$ is inferred from an inter-arrival time $\delta' = t_{i+3} - t_{i+1}$ that corresponds to a deletion, it does not necessarily have an in-order match in the true rotation sequence S. Essentially, a deletion error causes a gap in synchronization of length 2. More precisely, a single deletion error that causes encoded bits b_{i+1} and b_{i+2} to be lost will generally have the property:

$$(v_i|\widetilde{v_i}) \approx s_i \text{ and } (v_{i+2}|\widetilde{v_{i+2}}) \approx s_{i+3} \text{ but } (v_{i+1}|\widetilde{v_{i+1}}) \not\approx s_{i+1} \text{ and } (v_{i+1}|\widetilde{v_{i+1}}) \not\approx s_{i+2}$$

There is a chance of a false negative, however, as a deletion error results in a δ' that causes $(v_{i+1}|\widetilde{v_{i+1}}) \approx s_{i+1}$. Since $(v_{i+2}|\widetilde{v_{i+2}}) \not\approx s_{i+2}$ still holds, there would still be a synchronization gap of length 2 following s_{i+2}. Therefore, a deletion error can still be detected. Another way of alleviating this issue is to choose the random stream such that a single deletion error does not cause $(v_{i+1}|\widetilde{v_{i+1}}) \approx s_{i+1}$. More so, this analysis can occur offline, post-collection, so comparisons between observed and calculated sequences can be performed on a large sequence of elements to eliminate false negatives.

Multiple consecutive deletion errors can be detected based on a similar principle. In this case, the gap in synchronization is greater than length 2. For j consecutive deletion errors, causing bits $b_{i+1} \dots b_{i+1+j}$ to be lost, we have:

$$(v_i|\widetilde{v_i}) \approx s_i \text{ and } (v_{i+1}|\widetilde{v_{i+1}}) \approx s_{i+2+j} \text{ but } (v_{i+1}|\widetilde{v_{i+1}}) \not\approx s_{i+1+k} \quad \forall k \text{ where } k \in \{1 \dots j\}$$

Example: To better illustrate our synchronization error correction algorithm, it is useful to look at a simple example. Suppose the covert channel sender wishes to send the bit sequence $\{1, 0, 0, 0, 1, 1\}$, and the parties have decided on a pseudo-random sequence $\{20, 50, 30, 70, 40, 30\}$ and a window size of 100ms. At the covert sender's end, the modified inter-arrival times are (modulo the window size) $\{70, 50, 30, 70, 90, 80\}$. Suppose a deletion error occurs at the covert channel sender's end causing bits b_3 and b_4 to be lost. The inferred rotations V seen by the receiver is now: $\{(20|70), (50|0), (40|90), (40|90), (30|80)\}$. Note that the third inferred rotation is $(40|90)$ due to the concatenation of inter-arrival times of the encoding for bits b_3 and b_4.

However, based on the knowledge of the shared rotation sequence, S, the receiver expects: $\{20, 50, 30,$
$70, 40, 30\}$. The covert channel receiver recognizes a gap in synchronization between V and S at index 3 (elements 30 and 70 are missing) with an additional spurious element ($(40|90)$) caused by the concatenation of the two inter-arrival times. Once such a gap in matching is detected, the receiver suspects that a deletion error has occurred and marks the missing bits. Finally, the decoded bit sequence at the covert channel receiver is $\{1, 0, ?, ?, 0, 1, 1\}$ where $?$ denotes bits missed due to deletion errors.

Synchronization with Network Jitter. With network jitter added to the mix, exact matching between the inferred and true rotation sequences can no longer be performed. Network jitter may cause the inferred rotations to deviate from their true values, but no packets, and consequently no inter-arrival times, are actually lost. Additionally, this inexact matching factors out continuous random delay c.

Example: The covert sender is again using the rotation sequence $\{20, 50, 30, 70, 40, 30\}$, but this time network jitter is present. Assume that the network jitter is of the order of $10ms$, a fairly conservative value for most paths on the Internet, the covert channel receiver's inferred rotations might be: $\{(23|73), (43|93),$
$(35|85), (70|20), (44|94), (30|80)\}$.

By allowing for inexact matching of the inferred rotation sequence with that of the true derived rotation sequence, the covert channel receiver notices that the observed rotation values are very close to the expected values within the 10ms range. Moreover, the covert receiver notices that the sequences are in sync at $(70|20)$ and $(30|80)$ and is abled to definitively say that the deviations were caused by network jitter rather than an insertion or a deletion error. In the presence of errors, a combination of sequence alignment and inexact matching is performed.

Formally, in the presence of network jitter, with j consecutive deletion errors, causing bits $b_{i+1} \dots b_{i+1+j}$ to be lost, we have:

$$(v_i|\widetilde{v_i}) \approx s_i \;\; (v_{i+1}|\widetilde{v_{i+1}}) \approx s_{i+2+j} \text{ but } (v_{i+1}|\widetilde{v_{i+1}}) \not\approx s_{i+1+k} \quad \forall k \text{ where } k \in \{1 \dots j\}$$

where $(v|\widetilde{v}) \approx s$ if:

$$s - \varepsilon < v < s + \varepsilon \quad \text{or} \quad s + \varepsilon < \widetilde{v} < s + \varepsilon \qquad \varepsilon = \text{Inexact matching threshold}$$

The parameter ε decides the aggressiveness with which matching is performed and is chosen based on the expected network jitter. However, setting the value too large may cause the covert receiver to miss deletion errors (false negatives). On the other hand, if the value is too small, the receiver might falsely detect deletion errors (false positives) causing the rotation sequence to go out of sync.

Note, the inexact matching parameter, ε, differs from the error bounding parameter, ϵ. Although they are both used to handle network jitter, the first is solely for the purpose of synchronization, and the latter is used solely for decoding. There is no requirement that $\varepsilon = \epsilon$, and in fact, we found that small values of ε (on the order of 3% of the number of rotation splits) are sufficient while larger values of ϵ (on the order of 33% of the window size) enhance the decoding performance.

Finally, under certain conditions, this encoding scheme also allows for partial recovery when deletion errors occur. Additional details are found in the Appendix.

6 Detection using Entropy

Gianvecchio and Wang [8] propose using the first-order entropy of packet inter-arrival times to detect the JitterBug timing channel. Below, we describe the technique, the detection model and the evaluation methodology as presented in their work.

Entropy detection attempts to measure the relative similarity between legitimate traffic and suspected covert channel traffic. Since timing channel encoding schemes embed information within network timings by modifying inter-packet delays, they affect the distribution of inter-arrival times. The detection technique relies on observing such a change in distribution using an entropy heuristic. The entropy calculation is performed by assuming a distribution from a previously sampled known legitimate traffic. Any deviation from this training distribution causes a drop in the calculated entropy, and can be used as an indicator of a timing channel.

Calculating the entropy using a finite number of timing samples requires the use of an empirical probability distribution. This is estimated using the method of histograms. Data is binned into Q bins, where Q is the bin granularity. For the JitterBug, a fine-grained equiprobable binning strategy (with $Q = 65536$) is employed.

The entropy threshold is calculated as follows. First, the bin ranges and distributions are calculated using legitimate timing samples from the training set. Given this binning distribution, the first-order entropy can be calculated on groups of inter-packet delays from the same set. As we are interested in finding a lower limit on the entropy of legitimate traffic, a cutoff score can be chosen based on a desired false positive rate. For example, if the target false positive rate is chosen as 0.01, then the 99th percentile calculated entropy score from the training set is chosen as the threshold.

Detection is achieved by sampling suspected inter-arrival times using the same sample size used during threshold calculation. When the calculated entropy of the suspected sample falls below the threshold entropy, it is classified as containing a covert channel. The intuition behind this approach is that if the observed timing samples have a different distribution compared to the one used for binning, then their calculated entropy is decidedly lower.

To evaluate detection performance with the JitterBug, Gianvecchio and Wang use a data set consisting of 200,000 legitimate SSH traces. The binning distribution is estimated from these legitimate timing samples. For threshold calculation, entropy is calculated for 100 groups of 2000 samples with a target false positive of 0.01. The same sample set is then reused to calculate JitterBugged timing samples using an encoding simulator. As presented in the work, entropy detection works very well with a zero false negative rate. However, as we describe below, the detection model and experimental methodology suffers from a few problems.

Limitations of the Entropy Detection Model. An important assumption when using measured entropy to detect a timing channel is that shifts in inter-arrival time distributions are indicative of the presence of a timing channel. This assumption is not necessarily true because it depends on the network setup as well as the placement of the detector and encoder. In particular, even if network timings from a single host are observed, there can be high variance amongst timing distribution for different applications and different users. We demonstrate this phenomenon in Section 7.3 and Section 7.4. These slight variations can result in significantly high false positive rates. Moreover, the precise training set used for threshold calculation and, in particular, the sample set used to calculate the binning distribution, has a significant impact on the effectiveness of entropy rate detection.

Limitations of Previous Detection Methodology. Gianvecchio and Wang's work claims a very low false negative rate for detecting JitterBug. In practice, the binning distribution and the entropy threshold must be calculated a priori using previously observed (and legitimate) timing samples. However, as presented in previous work, this distinction is not clearly maintained. In particular, the evaluation of entropy detection is performed by using covert timing samples derived from the training set itself.

7 Experimental Results

We experimentally evaluate the steganographic timing channel for its resistance to standard detection techniques and practical performance. In the interest of performing "worst case" experiments for detection resistance, we allow the detector access to timing samples output by the steganographic encoder without any added noise (e.g. network jitter). This effectively simulates a detector at the source of the channel transmission, which is the most favorable position for the warden (any added network jitter masks the covert timing channel even further). Channel performance is evaluated using both offline (using simulated network jitter and errors) and online experiments (under actual Internet conditions).

We use two different datasets consisting of timing samples from actual network traffic. The first dataset consists of SSH keystroke traces collected from 3 users over a period of nine months. The traces consist of approximately 94,270, 157,840, and 133,160 keystroke timings per user, totaling more than 375,000 keystrokes during roughly 2000 individual SSH session. To perform the collection, we modified a SSH client to record the keystroke timings as the user typed in an interactive SSH session. During the collection period, each user had different workloads and usage patterns leading to varied and unique timing sample distributions.

The second dataset consists of timings of HTTP GET Requests. These samples are derived from the IRCache Squid Proxy HTTP Traces [24] in the Internet Measurement Data Catalog [1]. A timing channel within HTTP GET requests assumes a more powerful attack model when compared to the Keyboard Jitterbug because embedding must be performed by an intermediate network node, e.g., a proxy.

In all experiments we use a window size (w) of 20 milliseconds. This value was suggested in previous work [17] as giving good performance in the presence of network jitter effects while keeping added delays small enough to be inconspicuous to users.

7.1 Methodology

To simulate a variety of real detection conditions, we measured several different scenarios. The experiments are performed using both SSH keystrokes as well as the timings of HTTP GET requests.

For SSH, we assume two different usages: a single user workstation and a shared, multi-user workstation. Each user's data set is split in half and training is performed using the first half. On the second half of the dataset, we ran detection tests on the original timing samples as well as the same samples with the steganographic encoding. This setup allows estimation of both false positive and false negative rates when training is performed *a priori*.

The datasets for a shared, multiple typist workstation are simulated in two ways. In the first scenario, the detector observes timings from simultaneous SSH sessions but is unable to distinguish between individual sessions. This also effectively simulates a workstation where three users are SSHing out simultaneously during the collection period. The second dataset interweaves individual sessions and the detector observes only isolated user timings at any given time. This simulates a shared workstation used by multiple users where only a single user is SSHing out at any given time.

Entropy Training. In all experiments we use a training methodology that mirrors previous work [8] (Section 6). Unlike Gianvecchio and Wang, we perform our experiments with an *a priori* distribution and binning analysis to compute a threshold value, and then evaluate the effectiveness of this threshold for detecting our steganographic encoding. All experiments calculate entropy based on the first-order entropy and not the higher-order corrected conditional entropy. As presented in [8], the detection rate of corrected conditional entropy is orders of magnitude less than that of first order entropy.

To compute the threshold, we train using half the dataset and on average, we trained on 50,000 keystroke timings for single user sessions, 150,000 timings for multi-user sessions, and 200,000 samples for HTTP GET Requests. To achieve a realistic detection model, we do not randomize the order of the sessions, i.e., we train on exactly the first half of the collection period. We choose thresholds based on a 2% false positive rate for single-user sessions, and

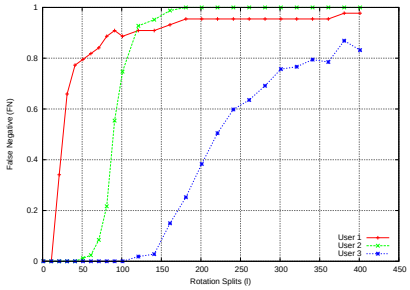

Figure 4: False negative ratio for various rotation splits using three different users.

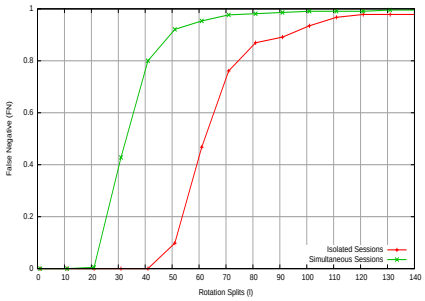

Figure 5: False negative ratio for using collated timing smaples during simultaneous and isolated sessions for three different users.

Training Set	Testing Set	FP Ratio
User 1	User 2	0.00
User 1	User 3	0.27
User 2	User 1	1.00
User 2	User 3	0.64
User 3	User 1	1.00
User 3	User 2	0.32

Table 1: False Positive (FP) ratios for different pairs of training and testing sets. Target false positive is set to 0.01.

Network Host	RTT	w/o Del.	w/Del.
ColumbiaU (New York, NY)	5 ms	99.0%	97.2%
UKansas (Lawrence, KS)	43 ms	99.7%	96.8%
UTexas (Austin, TX)	52 ms	99.6%	94.4%
UUtah (Salt Lake City, UT)	67 ms	99.6%	96.4%
UCSD (San Diego, CA)	81 ms	99.8%	97.6%
ETHZ (ETH, Zurich)	111 ms	99.8%	95.0%

Table 2: Measured accuracy of the timing channel to various network hosts with and without simulated deletions. The values are averaged over 5 runs. Rotation Granularity is fixed at 1000 and $\varepsilon = 30$ or 4% of the number of splits.

we choose a 1% false positive rate for multi-user sessions and HTTP GET requests. The steganographic encoding is then performed on the latter half of the dataset to analyze its covertness.

7.2 SSH: Single Typist

In this scenario, the covert channel detector is assumed to be able to identify the source of the interactive network traffic and can therefore train based on the typing distribution from a single user. As individual users show regularity and patterns in their typing behavior, we expect entropy detection to work well in capturing the distribution of legitimate inter-packet delays. Figure 4 shows the false negative ratios for different granularities of rotations for 3 different users. As seen in this figure, higher values of rotation splits lead to higher false negatives.

We also notice a significant difference in performance across different users. In particular, for User 1, a rotation granularity of around 30 leads to a false negative ratio greater than 0.5. However, for User 3, the false negative ratio does not go above 0.5 until the number of splits is increased to 250. With a larger l, the shift in the distribution is not large enough to cause the entropy to decrease by a sufficient margin. Our results indicate that using entropy to characterize user typing behavior (i.e., legitimate SSH timing distribution) is not adequate to detect the steganographic timing channel.

7.3 SSH: Multiple Users

In this experiment, we evaluate entropy detection when the training set used to calculate the binning distribution and entropy threshold is from a different user than the one whose traffic is being monitored.

Different training and testing user. In this scenario, the detector is trained using samples from one user and tests are run on timing samples of a different user. Table 1 shows the false positive ratios for different pairs of training and testing samples. The entropy threshold is calculated by using a target false positive of 0.01. The false positive rates are very high, and thus, covert channel detection based on the entropy metric is unusable in this scenario. These results indicate that entropy detection is highly sensitive to the training dataset for different users using the same

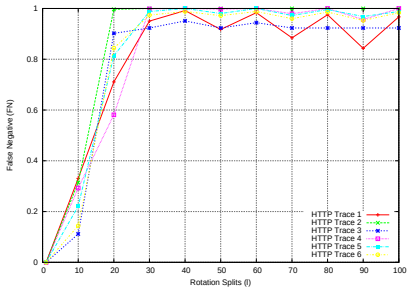

Figure 6: HTTP GET Requests: False Negative ratio for different rotation splits.

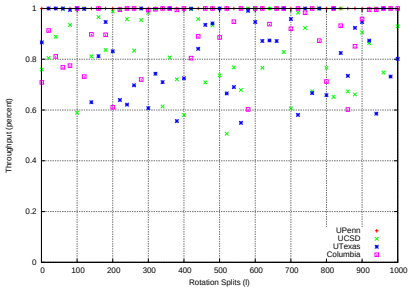

Figure 7: Measured accuracy for different rotation granularities using simulated jitter to network hosts.

network application. Therefore, the choice of training set can have a significant effect on the accuracy of entropy detection.

Note that during threshold calculation, the choice of target false positive ratio has a commensurate effect on the achieved false negative rate. Higher target false positive rates result in lower false negatives. However, this also causes the actual false positive rate (on the testing set) to be higher. The situation is even worse when the testing and training timing samples are derived from different users or network applications as shown above. As the target false positive rate is increased, the observed false negative goes down. But this improvement in false negative rates comes at the expense of a higher observed false positive rate.

Using an aggregate training set. In practice, the covert channel detector lies somewhere near the network boundary where it can observe network timings of outgoing traffic. For example, the detector could look at all outgoing interactive SSH sessions for suspected timing channel embeddings. Our experiments in Section 7.3 showed that the distribution of timing samples exhibit a large variation amongst different users even for the same network application.

One way of avoiding this problem is for the detector to train based on aggregate data collected from multiple users or network applications. From a practical standpoint, the network monitor performing entropy detection may not be able to distinguish network traffic emanating from the same host for different users. Since the training is performed *a priori*, using an aggregate training set is the only practical way of performing covert channel detection in such a scenario.

Figure 5 shows the performance of entropy detection when training is performed using aggregate timing data from three different SSH users in isolated or simultaneous sessions. Isolated session simulate a shared workstation where only a single user is SSHing out at any given time. Simultaneous sessions simulate two scenarios; a shared workstation where all three users may be SSHing out at the same time, or a detector that observes all SSH packets without differentiating between their source.

7.4 HTTP GET Requests

With HTTP timing traces, entropy detection fares even worse. A covert channel encoding on HTTP GET requests timing assumes a more powerful attack model where the attacker controls an intermediate node along the path, e.g., a proxy. The attacker intercepts HTTP traffic and delays certain HTTP requests to embed a covert timing channel.

Figure 6 shows the performance of entropy detection on HTTP GET request timings of 6 different sites. As seen above, entropy detection performs even worse with HTTP traffic than with interactive SSH sessions. With an increased rotation granularity, the false negative ratio is very high for relatively low number of splits. This is primarily due to the fact that HTTP requests do not necessarily have a stable distribution compared to interactive SSH sessions where it is largely a function of user typing behavior. Therefore, an entropy metric is not sufficiently discriminating in distinguishing between legitimate HTTP traffic and one with our steganographic timing channel.

7.5 Decoding Performance

We evaluate the performance of steganographic timing channels under both simulated and real network conditions. The bandwidth of the channel is one bit per network packet. However, the decoder must contend with the effects of

network jitter as well as synchronization errors due to insertions or deletions.

Effect of Rotation Granularity. For analyzing the effect of different rotation granularities on performance, we use simulated network jitter. Artificial jitter is added to covert traffic by modelling network effects to different host on the Internet. We measure jitter statistics to a particular host, and delay packets based on a Gaussian distribution. Note that even though network jitter does not necessarily exhibit a Gaussian distribution, this experiment allows us to measure any correlations between the granularity of rotation and channel performance under network jitter.

Figure 7 shows the measured timing channel accuracy for different rotation granularities. Interestingly, rotation granularity does not significantly impact the performance of the timing channel under network jitter. Therefore, the steganographic timing channel can use higher rotation granularities which should help synchronization performance in the presence of insertion and deletion errors.

Real Network Conditions. For a practical evaluation of performance, we also test the steganographic timing channel under real network conditions. We initiate SSH sessions to various hosts on the PlanetLab network [7]. An SSH replayer simulates keyboard activity over these connections at the timing intervals derived from the covert embedded timestamps, effectively simulating a steganographic encoding being used by a Keyboard JitterBug. The decoder runs at the target host and uses the timing of incoming SSH packets to decode information from the timing channel. Since rotation granularity does not sufficiently impact channel performance, we choose to use a rotation granularity of $l = 1000$ and a decoding error bound ϵ to $w/3$.

Table 2 shows the measured accuracy of the steganographic timing channel for various Internet hosts. All SSH connections are initiated from our lab network and measurements are made at the destination node. We use a window size of 20ms, and even with a very high rotation granularity, the decoding accuracy is very high.

Effect of Synchronization Errors. To test the effect of synchronization errors, in particular deletion errors, we simulated deletions using the same experimental setup as before. At the sender, we randomly drop packets with a probability of 2% after encoding but before sending them over the network. This effectively simulates deletion errors due to OS-level buffering. Our choice of drop rate is very conservative compared to our empirical observations of deletions for interactive SSH sessions where such drops are very rare. We perform our tests only with deletion errors as insertion errors are handled using the same mechanisms.

At the receiver's end, the decoder uses a synchronization algorithm based on the principles described in Section 5. Our algorithm identifies deletion (and insertion) errors and resynchronizes the observed rotations to the true ones. It works using an extension of the Smith-Waterman string alignment algorithm [20]. By aligning the expected rotations with the observed rotations, the algorithm determines a best fit, then inserts synchronizing timestamps to make up for deletion errors (or remove them for insertions). High network jitter may sometimes cause the decoder to wrongly assume that a deletion occurred due to a match failure. However, the algorithm handles these false positives by reverting the added synchronizing timestamps when the resynchronization occurs at some later point during alignment.

Table 2 also shows the measured accuracy in the presence of isolated deletion errors. We use inexact matching with $\varepsilon = 30$, i.e, a 3% error is allowed while matching elements in the inferred rotation to the expected rotation sequence. We found that this value is sufficient in handling the variations in network delays over the Internet. Although the above results correspond to isolated deletion errors, the algorithm also effectively deals with bursty deletion errors in the same way.

8 Conclusion

In this paper, we described a practical steganographic timing channel. We identified two key properties for a steganographic timing channel: it should be cryptographically keyed and the scheme should have a low distribution parity between input and output. Our scheme provides channel secrecy through the use of cryptographic primitives (a stream cipher). Without knowledge of the secret key, the channel contents cannot be effectively decoded nor detected. Although we cannot provide perfect input/output distribution parity, our scheme provides it in a restricted form through the covert uniformness property. Our steganographic encoding always outputs inter-arrival times that are uniformly distributed modulo a timing window. Consequently, input inter-arrival times that also meet this condition (as we expect keystroke timings do for reasonable values of w) will be indistinguishable from the output without knowledge of the cryptographic pseudo-random stream.

Our empirical study of the covertness of the scheme shows that the steganographic timing channel is resistant to detection through the best known practical detection heuristic. In particular, we show that the use of entropy detection suffers from many practical limitations. Most importantly, the inherent variation in timings of network traffic, even for the same application with different users, is sufficient enough to cause high false positives.

The steganographic encoding also performs well in practice. Under real network conditions, the decoding accuracy is similar to previously described schemes, and it performs well in the presence of errors. Moreover, the use of a shared rotation sequence not only provides enhanced covertness, but also allows for synchronization between the sender and receiver without a feedback mechanism. Insertion and deletion errors can be detected and handled, thus obviating the need for higher level mechanisms such as sequence numbers. Additionally, a high rotation granularity does not seem to affect decoding accuracy, so the scheme gains covertness without any measurable loss in performance.

References

[1] Internet data measurement catalog. http://imdc.datcat.org.

[2] M. Backes and C. Cachin. Public-key steganography with active attacks. In J. Killian, editor, *TCC 2005, LNCS*, volume 3378, pages 210–226. Springer, 2005.

[3] V. Berk, A. Giani, and G. Cybenko. Detection of Covert Channel Encoding in Network Packet Delays. Technical report, Darthmouth College, 2005.

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

[5] S. Cabuk, C. E. Brodley, and C. Shields. IP covert timing channels: design and detection. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 178–187, New York, NY, USA, 2004. ACM Press.

[6] C. Cachin. An information-theoretic model for steganography. *Information and Computation*, 192(1):41 – 56, 2004.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[8] S. Gianvecchio and H. Wang. Detecting Covert Timing Channels: An Entropy-Based Approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, 2007.

[9] N. J. Hopper, J. Langford, and L. von Ahn. Provable secure steganography. In M. Yung, editor, *CRYPTO 2002, LNCS*, volume 2442, pages 77–92. Springer, 2002.

[10] R. A. Kemmerer. A Practical Approach to Identifying Storage and Timing Channels. In *IEEE Symposium on Security and Privacy*, 1982.

[11] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, IX:5–83, 1883.

[12] X. Luo, E. Chan, and R. Chang. TCP covert timing channels: Design and detection. *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 420–429, June 2008.

[13] T. Mittelholzer. An information-theoretic approach to steganography and watermarking. In A. Pfitzmann, editor, *Information Hiding, 3rd international Worshop, LNCS*, volume 1768, pages 1–16. Springer, 1999.

[14] S. J. Murdoch and S. Lewis. Embedding Covert Channels into TCP/IP. In *Information Hiding Workshop*, 2005.

[15] F. A. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information Hiding – A Survey. *Proceeding of the IEEE, special issue on protection of multimedia context*, 1999.

[16] N. E. Proctor and P. G. Neumann. Architectural Implications of Covert Channels. In *15th National Computer Security Conference*, 1992.

[17] G. Shah, A. Molina, and M. Blaze. Keyboards and Covert Channels. *Proceedings of the 15th Usenix Security Symposium*, August 2006.

[18] G. Simmons. The prisoner's problem and the subliminal channel. *Proceedings of CRYPTO '83*, 1984.

[19] R. Smith and G. Scott Knight. Predictable design of network-based covert communication systems. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 311–321, May 2008.

[20] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[21] L. von Ahn and N. J. Hopper. Public key steganography. In C. Cachin and J. Camenisch, editors, *Advances in Cryptography: Eurocrypt 2004, LNCS*, volume 3027, pages 322–339. Springer, 2004.

[22] X. Wang, S. Chen, and S. Jajodia. Tracking anonymous peer-to-peer VoIP calls on the internet. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 81–91, New York, NY, USA, 2005. ACM Press.

[23] X. Wang, S. Chen, and S. Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 116–130, May 2007.

[24] D. Wessels. "IRCache traces for DITL January, 2007 (collection)". http://imdc.datcat.org/collection/1-01J0-5=IRCache+traces+for+DITL+January%2C+2007 (accessed on 2009-03-30).

[25] S. Zander, G. Armitage, and P. Branch. Survey of covert channles and countermeasures in computer network protocols. In *IEEE Communications Surveys*, volume 9, pages 44–57. 2007.

[26] J. Zölner, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf. Modeling the security of steganographic systems. In D. Aucsmith, editor, *Information Hiding, 2nd international Worshop, LNCS*, volume 1525, pages 344–354. Springer, 1983.

A General Correction

Another property of the rotation and synchronization algorithm is that under certain conditions, it allows for partial recovery of the encoded sequence when a deletion error occurs. From Figure 3, it is clear that $\delta' = t_{i+3} - t_{i+1} = \delta_{i+1} + \delta_{i+2}$. We have:

$b_{i+1} = \mathcal{D}(\delta_{i+1}, s_{i+1})$
$b_{i+2} = \mathcal{D}(\delta_{i+2}, s_{i+2})$
$b' = \mathcal{D}(\delta_{i+1} + \delta_{i+2}, s_{i+1} + s_{i+2}) = \mathcal{D}(\delta', s_{i+1} + s_{i+2})$
Now,
$\delta' = \mathcal{E}(\{0,1\}, s_{i+1}) + \mathcal{E}(\{0,1\}, s_{i+2})$
And so,
if $b_{i+1} \oplus b_{i+2} = 0,$ then $(\delta' - s_{i+1} - s_{i+2}) \bmod w = 0$
if $b_{i+1} \oplus b_{i+2} = 1,$ then $(\delta' - s_{i+1} - s_{i+2}) \bmod w = \lfloor w/2 \rfloor$
Therefore,
if $b' = 0$, then $b_{i+1} \oplus b_{i+2} = 0$
if $b' = 1$, then $b_{i+1} \oplus b_{i+2} = 1$
Where,
$\mathcal{D}$ is the decoding algorithm,
$\mathcal{E}$ is the encoding algorithm,
$\oplus$ is the bit-wise XOR

The above property holds because if two inter-arrival times are both 0 or both $w/2$ modulo the window size, then their sum will always be 0 modulo the window size. Similarly, if one inter-arrival time is 0 and the other $w/2$ modulo the window size, then their sum will be $w/2$ modulo the window size. This property allows us to guess the value of the missing bits with a probability better then guessing by chance individually for each lost bit.