On Blending Attacks For Mixes with Memory Extended Version

Luke O'Connor *

Abstract

Blending attacks are a general class of traffic-based attacks, exemplified by the $(n-1)$-attack. Adding memory or pools to mixes mitigates against such attacks, however there are few known quantitative results concerning the effect of pools on blending attacks. In this paper we give a precise analysis of the number of rounds required to perform a blending attack for the pool mix, timed pool mix, timed dynamic pool mix and the binomial mix.

1 Introduction

Mixes, first proposed by Chaum [3], are a means of providing unlinkability between a set of messages received and subsequently forwarded by the mix. This is typically achieved through a combination of cryptographic techniques and traffic manipulation such as delaying, padding and re-ordering. The original proposal of Chaum, referred to as a *threshold mix*, has been extended and improved by many authors (see [12, 11, 7] for surveys). Anonymous remailer systems such as Mixmaster [10], and its successor Mixminion [4], are embodiments of (and improvements to) the principles originally presented by Chaum. Anonymity systems are subject to a wide variety of threats including attacks pertaining to replay, blending, pseudospoofing, tagging, intersection and timing [4]. The threat that such attacks pose against a given anonymity system will depend on the system design and its operational characteristics, as well as the control that an attacker can can exert on the mix or its environment to mount the attack.

In this paper we will be mainly concerned with *blending attacks*, which refer to a general class of attacks based on manipulating network traffic to compromise the anonymity of one or several messages sent to a mix [12]. The attacker is assumed to be a *global*

*This work was completed in February 2005 while the author was at IBM Zurich Research Laboratory, Switzerland. He can now be contacted via email at `luke.oconnor@swissonline.ch`.

active attacker, who is able to monitor all communication links, delete and delay legitimate message traffic, and insert arbitrary amounts of spurious traffic. The best known blending attack is the $(n-1)$-*attack*, where the traffic to a mix is manipulated so that it contains a message batch of size n that consists of a target message m^* and $n-1$ spurious messages generated by the attacker. When the mix is a standard threshold mix, the target message is guaranteed to be included in the next set of flushed messages, and can therefore be isolated and traced by the attacker. For standard threshold mixes, the $(n-1)$-attack requires just two rounds: one round to wait for the current mix contents to be flushed, and then one additional round to flush the target message while blocking other legitimate message traffic. The $(n-1)$-attack can always be attempted given a sufficiently powerful attacker, and the authors of [2] remark that there is "no general applicable method to prevent this attack".

A general strategy that mitigates against blending attacks is to introduce memory into a mix, referred to as the *pool*. A pool permits messages to be retained in the mix for several rounds before being forwarded, which increases the message delay but also mixes messages within larger message sets. With respect to the $(n-1)$-attack, the attacker must then first replace the current pool of legitimate messages in the mix with spurious messages, then submit the target message, and finally keep submitting more spurious messages until the target message is flushed. Therefore when a mix is equipped with a pool, the attacker will be required to control the mix for more rounds to arrange the circumstances for a successful $(n-1)$-attack as compared to a threshold mix.

If a mix designer is to validate their choice a system parameters (such as the batch size, pool size, or timing threshold) then a thorough understanding of the trade-off between these parameters and their influence on the success of blending attacks is required. For example, the timed dynamic pool mix is used in the recent design of the Mixminion remailer [4], and while the authors state that their batching strategy will force an attacker to spend multiple time intervals to complete an $(n-1)$-attack, no specific analysis is presented. The main reference on analysing blending attacks is [12], later revised and extended in [11]. While many cogent remarks concerning mix designs are made in [12, 11], accurate expressions for the number of rounds required to complete the $(n-1)$-attack on the basic threshold pool mix and the timed dynamic pool mix are not given.

Our main result is to derive the probability distribution for the number of rounds required to complete an $(n-1)$-attack against several types of mixes with memory. In particular we consider the pool mix, the timed pool mix, the timed dynamic pool mix and the binomial mix [12, 8]. Our analysis is based on a steady-state strategy where the attacker keeps the number of messages entering and leaving the mix constant over the rounds of the attack, which is true of the pool mix by design.

An outline of this paper can be given as follows. In §2 we introduce basic concepts and notations for mixes with memory, used throughout the paper. In §3 we give an overview of blending attacks, and then elaborate on the steady-state strategy for blending attacks

in §3.1. The basic threshold pool mix is analyzed in §4, and that analysis is then extended to the timed pool mix in §5, and the timed dynamic pool mix in §6. Finally, the binomial mix is discussed in §7, and our conclusions are presented in §8.

2 Mixes with Memory

As with memoryless mixes, mixes with memory operate over a series of *rounds*. At the beginning of round $r \geq 1$, the mix contains a message pool P_r of size $|P_r|$, that consists of the messages retained in the mix from round $r-1$. It is assumed that P_1 consists of dummy messages generated by the pool mix itself, denoted by the set B_0. The set of messages collected by the mix over the course of round r will be denoted by B_r, referred to as r-th batch set. Round r is terminated when the *batching condition* is satisfied. Let $S_r = (P_r \cup B_r)$ denote the set of messages resident in the mix when the batching condition is satisfied, referred to as the *selection set*. The mix then selects a subset F_r of messages from S_r to be flushed (forwarded), according to a *flushing rule*, referred to as the *flush set* F_r. Each message $m \in F_r$ is flushed, and the pool for the next round is then defined as $P_{r+1} = (S_r - F_r)$. The batching condition and flushing rule are collectively referred to as the *batching strategy* for the mix.

Mixes with memory can be parameterized to support a large variety of batching strategies (see [12, 11, 7] for extensive surveys). A basic design principle is whether the batching condition is threshold-based, time-based, or potentially a combination of both. The batching condition of a threshold mix is satisfied when $|B_r| = n$, and n is referred to as the *threshold parameter*. For threshold mixes the round structure is then defined as intervals over which the mix receives n messages. On the other hand, the flushing condition of time-based mixes is satisifed every t time units, where t is called the *mixing interval*. In this case the round structure is defined as the series of mixing intervals of length t. Another basic design distinction is whether the pool size is static or dynamic. For mixes with static pools, $|P_r| = N$ for all $r \geq 0$, while for dynamic pools, $|P_r|$ is a function of $|S_{r-1}|$, the number of messages in the mix at time $r-1$. These two properties can be combined to produce different variants on mixes mixes with memory including the pool mix, the timed pool mix, the threshold or timed pool mix, and so on. We also assume that in practice each mix has two additional integer parameters, $N_{\min}$ and $N_{\max}$, which handle boundary cases. When $|B_r| < N_{\min}$ the mix retains the current batch in the pool but does not flush any messages. On the other hand, when the mix receives a large number of messages it restricts the size of the batch to be less than $N_{\max}$ to avoid overflow conditions.

3 Blending Attacks

Blending attacks refer to a class of attacks where an attacker is able to manipulate the messages entering the mix, typically through through a combination of trickling and flooding operations [12]. The $(n-1)$-attack can be generalized to mixes with memory, and consists of two phases: *pool flushing* and *target flushing*. The object of pool flushing is to replace the current pool set P_r by spurious messages generated by the attacker. Once P_r has been flushed, the attacker can submit the target message m^*, and then submit additional spurious messages until the target message is flushed.

We note that the attacker can verify when the pool flushing is complete if they have knowledge of the internal parameters of the mix. For example, assume that the attacker knows the size N of the pool in a threshold pool mix. The attacker can then block legitimate traffic and submit batches of spurious messages until the number of the observed difference between the number of messages sent to, and flushed by, the mix is N.

For a typical parameter selection, the number of rounds required to complete the attack is dominated by the pool flushing, since flushing the target message from the mix can be viewed as special case of flushing a pool set of size 1. Our analysis therefore concentrates on the number of rounds required to flush a general pool size of $N \geq 1$.

3.1 The Steady-State Strategy

Various strategies could be pursued by an attacker to flush the current pool from the mix. Our analysis will be with respect to a particular strategy that we will refer to as the *steady-state strategy* [1]. In this strategy the attacker observes and/or manipulates a mix for some number of rounds, say up to and including round $r-1$, where $|F_{r-1}|$ messages were flushed from the mix. Assume that the attacker wishes to flush P_r from the mix. The attacker then blocks all legitimate message traffic to the mix and submits k batches $B_r, B_{r+1}, \ldots, B_{r+k-1}$ of size $|F_{r-1}|$ to the mix, which forces $|F_{r+j}| = |F_{r-1}|$ for $0 \leq j < k$. The batching rule is satisfied at each of these rounds (as it was at round $r-1$) and the mix will either flush immediately or when the next time interval completes.

Let $\Delta(P_r)$ be the random variable which describes the minimum number of rounds required to flush the pool set P_r, assuming that the mix is in a steady-state during these rounds. If $\Delta(P_r) = k$, for some $k \geq 1$, then $P_r \cap P_{r+k} = \{\emptyset\}$, and $P_r \cap P_{r+j} \neq \{\emptyset\}$ for $1 \leq j < k$, In other words, at least one message $m_i \in P_r$ was retained in the mix until the beginning of round $r+k-1$, but all such remaining messages are then included in the set of flushed messages F_{r+k-1} for that round.

[1] The steady-state strategy was described in [12], and we introduce the term here only as a convenient shorthand.

We will determine $\Pr(\Delta(P_r) > k)$ for $k \geq 1$, where the probabilities are defined by choices of the flush set F_r at each round. Thus when $\Pr(\Delta(P_r) > k) < \epsilon$, for some suitably small ϵ, the attacker has the assurance that P_r was flushed from the mix in k rounds or less with probability $1 - \epsilon$. If $\Pr(\Delta(P_r) = k)$ is desired, then for $k > 1$ this probability can be determined via

$$\Pr(\Delta(P_r) = k) \;=\; \Pr(\Delta(P_r) > (k-1)) - \Pr(\Delta(P_r) > k). \tag{1}$$

Equation (1) cannot be use to determine $\Pr(\Delta(P_r) = 1)$, but this probability can typically be calculated directly from the context of the attack.

4 The Threshold Pool Mix

The threshold pool mix is defined by two parameters: the threshold n, and the pool size N. Each round r is defined by collecting a message batch B_r of size n such that the contents of the mix are then $S_r = (B_r \cup P_r)$. The flush set F_r is then selected as random subset of S_r of size n. The messages of F_r are forwarded and the pool for round $r + 1$ is set to $P_r = (S_r - F_r)$. The values of n and N are static over all rounds, so that for $r \geq 0$, $|B_r| = |F_r| = n$, $|S_r| = N + n$, $|P_r| = N$. In the remainder of this section we will use the term 'mix' to mean a threshold pool mix.

Following the discussion of §3.1, the threshold pool mix is in a steady-state by definition since the threshold and pool size are fixed. We now consider the number of rounds $\Delta(P_r)$ required to flush the pool P_r. The probability of flushing P_r in a single round is

$$\Pr(\Delta(P_r) = 1) \;=\; \frac{\binom{n}{n-N}}{\binom{N+n}{n}}. \tag{2}$$

For $k > 1$, we will calculate $\Pr(\Delta(P_r) > k)$ by decomposing this probability into events concerning the individual messages from P_r. If we let $P_r = \{\, m_1, m_2, \ldots, m_N \,\}$, then for $1 \leq i \leq N$, let $A_{i,k}$ denote the event that $m_i \in P_{r+k}$, meaning that m_i was not flushed from the mix after k consecutive rounds. We will write $A_{i,1}$ as A_i, which is the event that m_i is not flushed after one round. By definition

$$\Pr(\Delta(P_r) > k) \;=\; \Pr\left(\bigcup_{j=1}^{N} A_{i,k}\right) \tag{3}$$

and the union operation of the RHS of (3) may be expanded using the inclusion-exclusion principle (IEP) [9]. To apply the IEP, various intersections (joint events) between the $A_{i,k}$ events must be computed. Conveniently the intersections of these events exhibit considerable symmetry when the mix is in a steady-state.

Theorem 4.1 For $N, n, k \geq 1$,

$$\Pr(\Delta(P_r) > k) \;=\; \binom{N+n}{n}^{-k} \cdot \sum_{j=1}^{N} \binom{N}{j} \binom{N+n-j}{n}^{k} (-1)^{j+1}. \tag{4}$$

Proof. We first note that $A_1 A_2 \cdots A_j$ is the event that *at least* the j messages $m_1, m_2, \ldots, m_j$ are retained in the mix. The probability of this event is then

$$\Pr\left(A_1 A_2 \cdots A_j\right) \;=\; \frac{\binom{N+n-j}{n}}{\binom{N+n}{n}} \tag{5}$$

which only depends on j, and thus $\Pr(A_1 A_2 \cdots A_j) = \Pr(A_{i_1} A_{i_2} \cdots A_{i_j})$ for all choices $1 \leq i_1 < i_2 < \cdots < i_j \leq N$. Since the flush set is chosen uniformly and independently at each round, it follows that

$$\Pr(A_{i_1,k} A_{i_2,k} \cdots A_{i_j,k}) \;=\; \left(\Pr(A_1 A_2 \cdots A_j)\right)^k. \tag{6}$$

Now consider the following derivation of $\Pr(\Delta(P_r) > k)$

$$\begin{aligned}
\Pr(\Delta(P_r) > k) \;&=\; \Pr\left(\bigcup_{j=1}^{N} A_{i,k}\right) && (7)\\
&=\; \sum_{j=1}^{N} \sum_{1 \leq i_1 < \cdots < i_j \leq N} \Pr\left(A_{i_1,k} A_{i_2,k} \cdots A_{i_j,k}\right) \cdot (-1)^{j+1} && (8)\\
&=\; \sum_{j=1}^{N} \binom{N}{j} \cdot \left(\Pr\left(A_1 A_2 \cdots A_j\right)\right)^k \cdot (-1)^{j+1}. && (9)
\end{aligned}$$

Equation (8) is simply the IEP expansion of (7), and (9) is derived by simplifying (8) using (6). Finally, the theorem follows from substituting (5) into (9). □

Let S_j be the sum of the first j terms of the RHS of (4), for $1 \leq j \leq N$. The Bonferroni inequalities [9] state that for $1 \leq j < N$,

$$S_{j+1} \;\leq\; \Pr(\Delta(P_r) > k) \;\leq\; S_j. \tag{10}$$

Setting $j = 1$ yields that $S_2 \;\leq\; \Pr(\Delta(P_r) > k) \;\leq\; S_1$ where

$$S_1 \;=\; N \cdot \left(\frac{N}{N+n}\right)^k. \tag{11}$$

$$
\begin{aligned}
S_2 &= S_1 - \binom{N}{2} \cdot \left(\frac{N(N-1)}{(N+n)(N+n-1)} \right)^k, \\
&= S_1 \left(1 - \left(\frac{N-1}{2} \right) \cdot \left(\frac{N-1}{N+n-1} \right)^k \right). \qquad (12)
\end{aligned}
$$

Here $S_1 = N \cdot (\Pr(A_1))^k$ is well-known as Boole's inequality, and is often referred to as the union bound. We see that S_1 is a good approximation to $\Pr(\Delta(P_r) > k)$ as k increases, since the inner term of (12) is tending to 1 as a function of k.

Table 1 shows Theorem 4.1 evaluated for several choices of n and N. For example, when $n = 160, N = 100$ then with probability better than a half, more than 5 rounds will be required to flush the pool; but the probability that more 15 rounds are required is 0.596×10^{-4}. The values given in Table 1 were computed using Maple [1], and required about 20 seconds of computation time on a moderately fast laptop. Computing that $\Pr(\Delta(P_r) > 30) = 0.217 \times 10^{-7}$ for $n = 1000$ and $N = 800$ was also quickly determined. Thus a mix designer can easily evaluate the effect of various choices of n and N against the the number of rounds to flush a given pool P_r.

n	N	$k, \Pr(\Delta(P_r) > k)$				
100	80	(3, 0.999)	(5, 0.761)	$(10, 0.238 \times 10^{-1})$	$(15, 0.417 \times 10^{-3})$	$(15, 0.723 \times 10^{-5})$
100	160	(5, 0.999)	(10, 0.719)	$(20, 0.966 \times 10^{-2})$	$(30, 0.755 \times 10^{-4})$	$(40, 0.588 \times 10^{-6})$
160	100	(3, 0.999)	(5, 0.575)	$(10, 0.706 \times 10^{-2})$	$(15, 0.596 \times 10^{-4})$	$(20, 0.501 \times 10^{-6})$
200	160	(5, 0.943)	(10, 0.047)	$(15, 0.834 \times 10^{-3})$	$(20, 0.145 \times 10^{-4})$	$(25, 0.250 \times 10^{-6})$

Table 1: Example bounds on $\Pr(\Delta(P_r) > k)$ derived from Theorem 4.1.

We recall that the $(n-1)$-attack has two phases: flushing the current pool P_r, followed by submitting and flushing the target message m^*. For a mix with given parameters n' and N', we simply apply Theorem 4.1 with $N = 1$, and $n = n' + N' - 1$ to determine the number of rounds required to flush m^*.

Example 4.1 *Consider a threshold pool mix with threshold $n = 160$ and a pool size $N = 100$. If an attacker undertakes steady-state strategy to flush the pool P_r at some round r, then from Table 1 this will require less than 15 rounds with probability $1 - 0.596 \times 10^{-4}$, and less than 20 rounds with probability $1 - 0.501 \times 10^{-6}$. If the attacker knows the value of N then they can verify the success of the pool flushing by submitting spurious batches until the difference between the number of message submitted and the flushed is N. There is less than 1 chance in a million that this phase will require more than 20 rounds. Once the pool has been flushed the attacker then submits the target message m^*, along with another batch of $n - 1$ spurious messages. The number of rounds required to flush the*

target message is given by Theorem 4.1 after setting $N = 1$ *and* $n = 160 + 100 - 1$*, which corresponds to flushing a pool of size 1. Again the success of this phase can be verified by the attacker. Evaluating Theorem 4.1 with these parameters shows that 3 or less rounds are required to flush the target message with probability* $1 - 0.531 \times 10^{-7}$*. Thus the attack can be undertaken is 23 rounds or less with high probability.* □

Undertaking an attack for 23 rounds may seem unrealistic, however since the mix is threshold-based, the attacker is able to flush the mix at will by submitting batches of size n. Such a lengthy attack would be more difficult to mount on a timed pool mix.

4.1 Estimating the Pool Size

Theorem 4.1 is derived assuming a knowledge of n and N, which will be true for the system designer, but may not be the case for the attacker. Of course, n will always be known since each flush set F_r is of size n, which can be passively observed by the attacker.

NEW. It can be shown that $\Pr(\Delta(P_r) > k)$ is a monotonically increasing function of N for fixed k, such that for any $\delta > 0$

$$\Pr(\Delta(P_r) > k \mid N) \quad \leq \quad \Pr(\Delta(P_r) > k \mid N + \delta). \tag{13}$$

So if the attacker does not know the exact value of the current pool, Theorem 4.1 can always upper bound $\Pr(\Delta(P_r) > k)$ by overestimating N.

However, the attacker can obtain very good estimates of N by 'sampling' the behaviour of the mix. The probability that a given message $m \in S_r$ is included in F_r at round r is easily shown to be $p = n/(N + n)$. Since n is known, the attacker can approximate the pool size given a sufficiently accurate estimator $\hat{p}$ for p. Assume that the attacker submits one (distinguished) message m_i to the mix for s rounds, $1 \leq i \leq s$. Let $X_i = 1$ if m_i was flushed at round i, and let $X_i = 0$ otherwise. Then $\hat{p} = (\sum_{i=1}^{s} X_i)/s$ is a normally distributed estimator of p.

Example 4.2 *Let* $n = 160$ *and let* $N = 100$*. We have used Maple to generate 500 random bernoulli samples with parameter* $p = 160/(100 + 160) = 0.6154$*, of which 318 were successes, yielding that* $\hat{p} = 318/500 = 0.636$*. This corresponds to an attacker submitting 500 messages to the mix in 500 different rounds, and observing that a message was flushed at the same round it was received with probability* 0.636*. Using the normal distribution, the true value of* p *lies in the interval* $0.594 \leq p \leq 0.637$ *with 95% confidence. Solving for* N *given that* $p = n/(N + n)$ *yields that* N *lies in the range* $109.42 \leq N \leq 91.216$ *with 95% confidence. Increasing the number of samples to 1000 yields improves the interval to* $106.615 \leq N \leq 93.706$*.*

We note that it is relatively simple for the attacker to obtain a large number of samples from the mix since the attacker can passively obtain the samples. Since the mix is

threshold-based, samples can be obtained rapidly in times or high or even moderate traffic. Further since n and N are fixed parameters, the attacker can take the samples over a potentially long period and then later mount an $(n-1)$-attack using the steady-state strategy.

5 The Timed Pool Mix

In this variant of the pool mix, batches are defined by fixed mixing intervals of time t, with a fixed number of messages retained at each round. If the message batch B_r collected during the r-th time period is of size n_r, then P_{r+1} is selected as a random subset of size N from $S_r = (P_r \cup B_r)$, and the flush set is defined as $F_r = (S_r - P_{r+1})$. If $n_r = 0$ then $P_{r+1} = P_r$ and no messages are flushed. For any given round r, this construction is equivalent to the threshold pool mix with a threshold of $n = n_r$. Example parameters were given in [6, p.311], where the following rule ensures that $N = 20$: if $n_r \leq 20$ then $|F_r| = 0$; otherwise $|F_r| = (1 - 20/n_r) \cdot |S_r|$.

The steady-state strategy for the attacker is similar to the threshold pool mix, except that the attacker is able to increase the batch size in an effort to reduce the number of required flushing rounds. In practice the attacker cannot increase the batch size arbitrarily and the mix will have an internal limit N_{max} on the number of messages that the mix can hold. In any case, N_{max} may still be large enough relative to N so that the current pool can be flushed quickly. Since the pool size N is fixed, we then assume that $N_{\max}$ can be written as $N_{\max} = N + n_{\max}$ where $n_{\max}$ is the maximum batch size. We now outline a design strategy which mitigates against the attacker attempting to flood the mix and quickly flush the current pool.

5.1 A Design Strategy

Consider the following strategy for selecting the pool size N. The designer begins by selecting a maximum batch size $n_{\max}$ that is to be accepted at each round, perhaps based on expected traffic statistics. Let us assume that the mix is also designed so that with high probability an attacker blocking all legitimate traffic can be detected in at most k consecutive intervals with high probability, say $1-\epsilon$, for some appropriately small choice of ϵ. The system may be designed, for example, using the heartbeat traffic suggested in [5]. Given, $n_{\max}, k$ and ϵ the designer can now use Theorem 1 to find a pool size N so that $\Pr(\Delta(P_r) > k) > (1-\epsilon)$. The analysis assumes that the attacker is submitting $n_{\max}$ messages per batch in the stead-state strategy. But if the attacker actually submits less messages, then $\Pr(\Delta(P_r) > k)$ upper bounds the success of flushing the pool.

Example 5.1 *As an example, assume that the designer decides that the mix should accept no more than* $n_{\max} = 500$ *messages per interval, and that the system can (should!)*

confidently detect steady-state attacks after 3 rounds with probability $1 - 10^{-4}$*. Then using Theorem 4.1, selecting a batch of size* $N = 250$*, guarantees that for the steady-state strategy* $Pr(\Delta(P_r) > 3) > (1 - 10^{-4})$*. Further increasing* N *to 270 ensures that* $Pr(\Delta(P_r) > 3) > (1 - 10^{-5})$*. On the other hand, if the maximum batch size is increased to* $n_{\max} = 1200$*, then Theorem 4.1 yields that* $Pr(\Delta(P_r) > 3) > (1 - 10^{-4})$ *when* $N = 450$ *and* $Pr(\Delta(P_r) > 3) > (1 - 10^{-5})$ *when* $N = 480$*.* □

If the mix can accept large batch sizes relative to N, then the attacker may be able to flush the current pool before the attack is detected by the mix. The approach outlined above mitigates against this possibility by starting with an upper limit in the batch size to be accepted, and then deriving a pool size N which defeats the steady-state strategy for a given number of intervals k with a given probability $1 - \epsilon$.

The steady-state strategy for the attacker is similar to the pool mix except that the attacker has more freedom to choose the size n of the batch set (subject to $N_{\max}$ or an estimate of this parameter). The exact choice involves a tradeoff between the number of spurious messages and the number of time intervals to complete the attack, where in practice the latter consideration would seem to be of more importance. Indeed, the attacker can minimize the number of messages by keeping the mix in a steady-state through sending one spurious message per time interval until the pool is flushed (the attacker can verify when this occurs). Setting $n = 1$ in Theorem 4.1 and simplfying yields that

$$\Pr(\Delta(P_r) > k) \;=\; \sum_{j=1}^{N} \binom{N}{j} \left(1 - \frac{j}{N+1}\right)^k (-1)^{j+1}. \tag{14}$$

If $|P_r| = 100$ for example, then evaluating (14) shows that the attacker would require just over 500 single-message rounds to flush P_r with probability better than 50%. Thus the attacker is very likely to be detected, since all legitimate traffic must be blocked during the period of the attack.

If a given threshold is high relative to N, then P_r (or any of its remaining elements) are likely to be included in the current flush set. The probability that P_r is flushed after one time period is given in (2), which simplifies to $n_{\bar{N}}/(n+N)_{\bar{N}}$ where $a_{\bar{b}} = a(a-1)\cdots(a-b+1)$. This probability was erroneously reported in [12, §4.2] to be $N^2/(N+n)$, as expressed in our notation.

- comment on overestimating N ; - hypergeometric distribution population estimation

Alternatively, an attacker

6 The Timed Dynamic Pool Mix

The threshold and timed pool mixes considered previously have a pool size that is constant over each round. A dynamic pool mix determines the size of pool $|P_{r+1}|$ to be retained for the next round as a function of the number of messages $|S_r|$ currently in the mix. In this section we consider timed dynamic pool mixes, since these are of practical interest [10, 4]. Typically a dynamic pool mix is characterized by three parameters:

- The mixing interval t.
- The minimal size of the pool $N_{\min}$.
- The fraction $\alpha < 1$ of the messages to be flushed (subject to $N_{\min}$).

At the end of the r-th mixing interval, the mix considers its contents S_r. If $|S_r| \leq N_{\min}$ then $|F_r| = 0$ and no messages are flushed. Otherwise, let $|F_r| = \min(|S_r| - N_{\min}, \lfloor \alpha \cdot |S_r| \rfloor)$ and construct F_r as a random subset of S_r of size $|F_r|$.

Example 6.1 *An example parameterization of $\alpha = 0.65, N_{\min} = 45$ was considered in [7], which defines the following rules for the number of messages to be flushed:*

- *If* $0 < |S_r| \leq 45$ *then* $|F_r| = 0$*;*
- *If* $46 \leq |S_r| \leq 129$ *then* $|F_r| = |S_r| - 45$*;*
- *If* $|S_r| \geq 130$ *then* $|F_r| = \lfloor \alpha \cdot |S_r| \rfloor$.

Thus, when the mix contains 130 messages or more the pool size at the next round is then $|P_{r+1}| = |S_r| - \lfloor \alpha \cdot |S_r| \rfloor$. □

6.1 The Steady-State Strategy

We now derive the success of the steady-state strategy, assuming that the objective is to flush P_r and that the values of $N = |S_{r-1}|$ and α are known. In the next section we discuss how the attacker can determine a good estimate of α, and hence $|S_r|$ at any round. The attacker selects a round $r-1$ where the observed traffic has been sufficiently high to expect that $|F_{r-1}| = \lfloor \alpha \cdot |S_{r-1}| \rfloor$, or the attacker injects a moderate number of spurious messages to force this condition with high probability.

Theorem 6.1 If $|F_{r-1}| = \lfloor \alpha \cdot |S_{r-1}| \rfloor$ and $|S_{r-1} = N|$, then for $k \geq 1$,

$$\Pr(\Delta(P_r) > k) \;=\; \binom{N}{\lfloor \alpha N \rfloor}^{-k} \cdot \sum_{j=1}^{N - \lfloor \alpha N \rfloor} \binom{N - \lfloor \alpha N \rfloor}{j} \binom{N-j}{\lfloor \alpha N \rfloor}^{k} (-1)^{j+1}.$$

Proof. The proof is a direct adaption of Theorem 4.1 by observing that $|P_r| = N - \lfloor \alpha N \rfloor$ and

$$(\Pr(A_1 A_2 \cdots A_j))^k \;=\; \binom{N-j}{\lfloor \alpha N \rfloor}^k. \tag{15}$$

□

Example 6.2 *Let $\alpha = 0.6$, which is the default value for timed dynamic pool mix of Mixminion [4], and let there be $N = 200$ messages in the mix at round $r-1$. Then $|F_{r-1}| = \lfloor 0.6 \cdot 200 \rfloor = 120$. Given these parameters, $Pr(\Delta(P_r) > k)$ is calculated in Table 2 for several values of k.* □

α	N	$k, \Pr(\Delta(P_r) > k)$				
0.6	200	(3, 0.996)	(5, 0.556)	$(10, 0.835 \times 10^{-2})$	$(15, 0.859 \times 10^{-4})$	$(20, 0.879 \times 10^{-6})$

Table 2: Example bounds on $\Pr(\Delta(P_r) > k)$ derived from Theorem 6.1.

As with the time pool mix, the steady-state strategy may not have a sufficient number of rounds to succeed if the designer incorporates specific measure to detect the blocking of legitimate traffic.

6.2 Estimating the value of α

The value of α may not be know to the attacker, but a good estimate of α can be obtained follows. Assume that under normal operation the attacker observes that the mix at round r flushes $|F_r|$ messages where $|F_r| = \lfloor \alpha N \rfloor$ where $N = |S_r|$. If the attacker blocks legitimate messages during round $r+1$, and submits $2 \cdot |F_r|$ spurious messages then $|F_{r+1}| = \lfloor \alpha(N + \lfloor \alpha N \rfloor) \rfloor$. It follows that $|F_{r+1}|/|F_r|$ is a good approximation to $1 + \alpha$ since

$$\frac{|F_{r+1}|}{|F_r|} \;\approx\; \frac{\alpha(N + \alpha N)}{\alpha N} \;=\; 1 + \alpha. \tag{16}$$

If we define α^* as $F_{r+1}|/|F_r| - 1$ then $N^* = \lfloor |F_r|/\alpha^* \rfloor$ is a good approximation to N. For example, if $\alpha = 0.6$ and $N = 397$, then $\alpha^* = .600840336$ and $N^* = 396$. The attacker can now mount a steady-state attack using these estimates.

This (attack) method for estimating α assumes that $|F_r| = \alpha \cdot |S_r|$, and therefore that $(|S_r| - N_{\min}) > \alpha \cdot |S_r|$. In practice, we expect this condition to be satisfied when the mix contains as few as several hundred messages. The condition for the example rule given above with $\alpha = 0.65, N_{\min} = 45$ is 130 messages. If the attack is mounted during a period of low traffic to the mix then the attacker will need to send several hundred spurious messages to prime the mix for the attack.

6.3 Reducing the Pool Size

In the attack outlined above it may be the case that N is very large and it may be time consuming to keep refilling the mix with $\lfloor \alpha N \rfloor$ messages to maintain a steady-state. Since the attacker knows α or can estimate α, the current size of the mix can be closely approximated from observing a single flush. The attacker can then block all traffic into the mix and let the mix repeatedly flush, until N has been reduced to an reasonable value. After k such rounds the size of the mix has been reduced to approximately $N(1-\alpha)^k \approx Ne^{-k\alpha}$. We note that if F_r and F_{r+1} are two successive flush sets produced by this process then the ratio $\alpha^* = |F_{r+1}|/|F_r|$.

7 The Binomial Mix

In [8] Díaz and Serjantov propose a general model for mixes based on classifying flushing algorithms according the $|F_r|/|S_r|$, the fraction of messages flushed as compared to the total number of messages in the mix at round r. Let $f : N \to [0,1]$ denote the *flushing function*, such that if $f(|S_r|) = p$ then $|F_r|/|S_r|$. THE BINOMIAL MIX CAN BE THOUGHT OF A MULTI-POOL mix

The binomial mix was introduced in [8], and was further elaborated upon in [11, p.77]. The distinguishing property of these mixes is the use of a *flushing probability function* $f : \mathbb{N} \to [0,1]$ to determine the set of messages to be flushed. Binomial mixes are timed, and at the r-th interval f is evaluated to yield $f(|S_r|) = p_r$. Each message $m \in S_r$ is then included in F_r independently with probability p_r, and thus $|F_r|$ follows the binomial distribution with parameter p_r.

Potentially many functions could be used as a flushing probability function. In [8] the suggested function is the normal distribution[2]

$$\Phi_{\mu,\sigma}(s) \quad = \quad \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{s} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{17}$$

so that $p_r = \Phi_{\mu,\sigma}(|S_r|)$ at round r. No values for μ and σ were suggested in [8], but later in [11, p.80], there is graphical comparison between the Cottrell mix with $\alpha = 0.8$ and the binomial mix with $\mu = 100, \sigma = 40$. [3]

[2] In [8] the authors refer to the normal distribution as the cumulative normal distribution, but the former term is correct.

[3] Figure 5.6 [11, p.80] seems to be incorrect since it shows $\Phi_{100,40}(s)$ to be converging to 0.8, when in fact it must be converging to 1.

7.1 The Steady-State Strategy

We first derive a bound on the steady-state strategy, assuming that all relevant parameters are known. Consider a steady-state attack at round r such that $|F_{r-1}|$ was observed by the attacker, and let Let $p_{r-1} = f(|S_{r-1}|)$ and $N = |P_{r-1}|$. Adapting Theorem 7 with $\Pr(A_1 A_2 \cdots A_j))^k = (1-p_{r-1})^{jk}$, and letting $q = 1 - p_{r-1}$, yields

$$\begin{aligned}
\Pr(\Delta(P_r) > k) &= \sum_{j=1}^{N} \binom{N}{j} q^{jk} \cdot (-1)^{j+1} \\
&= -\sum_{j=1}^{N} \binom{N}{j} (-q^k)^j \\
&= -\left((1-q^k)^N - 1\right) \\
&= 1 - (1-q^k)^N, \qquad (18)
\end{aligned}$$

which simply states that $\Pr(\Delta(P_r) > k) = 1 - \Pr(\Delta(P_r) \leq k)$ when messages are selected for flushing independently. Also from (1) it follows that $\Pr(\Delta(P_r) = k) = (1-q^k)^N - (1-q^{k-1})^N$ for $k > 1$. Letting $k = \lceil \log_{\frac{1}{q}} N - \log_{\frac{1}{q}} \epsilon \rceil)$ for some suitably small $\epsilon < 1$, consider the following bound

$$\begin{aligned}
(1-q^k)^N &\leq \exp\left\{-Nq^k\right\} \\
&\leq \exp\left\{-q^{-\lceil \log_{1/q} \epsilon \rceil}\right\} \\
&\leq \exp\left\{-\epsilon\right\} \\
&= 1 - \epsilon + O(\epsilon^2), \qquad (19)
\end{aligned}$$

where we have used the bounds $\ln(1-x) < -x$ and $e^{-x} = 1 - x + O(x^2)$ for $x < 1$. It follows from (19) that

$$\Pr(\Delta(P_r) > k) = \epsilon + O(\epsilon^2). \qquad (20)$$

The $O(\epsilon^2)$ term can be improved to $\epsilon^2 \cdot e^{-\epsilon}/2$ since

$$e^{-\epsilon} = 1 - \epsilon + \frac{\epsilon^2}{2} \sum_{i \geq 0} (-\epsilon)^i / i! = 1 - \epsilon + \frac{\epsilon^2 \cdot e^{-\epsilon}}{2}.$$

The number of rounds k required for the steady-state strategy to succeed with the error bound given in (20) is then $\lceil \log_{\frac{1}{q}} (N/\epsilon) \rceil$.

7.2 The Robustness of Parameter Selection

In [8] it is stated that binomial mixes make blending attacks more probabilistic since the attacker becomes less certain of the number of messages which are retained in the mix from one round to the next. This is true, but the success of the steady-state strategy outlined above is quite robust against possible choices for N and $q = 1 - p_{r-1}$, as we now argue.

We first note that when $N \geq \mu$ then $p_{r-1} \geq \frac{1}{2}$, which for the example parameters above is satisfied when the mix has 100 messages or more. In this case, $q \leq \frac{1}{2}$ and $k = \lceil \log_2(N/\epsilon) \rceil$ steady-state rounds produce an error bound equal to or less than that of (20). So if the attacker overestimates N with $\hat{N}$, such that $\hat{N}/N = d$ then an additional $\lceil \log_2 d \rceil$ rounds will be added to achieve the error bound of (20) using $\hat{N}$, as compared to using the actual value of N. Thus overestimating N by a factor of 8 will only add 3 additional rounds to the attack as compared to using the exact value of N.

In this discussion we have assumed that $p_{r-1} \geq \frac{1}{2}$, which is true when $N = |S_{r-1}| \geq \mu$. Arranging this condition through flooding does not appear to be too difficult a task for for the attacker. No general theory for the optimal selection of μ and σ was presented in [8] or [11, p.77], but based on other mixes, we would expect μ to be less than 1000. Thus it seems quite feasible for the attacker to create a steady-state where $p_{r-1} \geq \frac{1}{2}$.

8 Conclusion

In this paper we have presented a general method for analysing the success of the $(n-1)$-attack using the steady-state strategy against a variety of mixes with memory (pools). This permits a mix designer to analyse parameters choices with respect to their effect on mitigating against the $(n-1)$-attack. Our analysis methods also permits an attacker to evaluate their success in undertaking a blending attack, assuming relevant parameters are known or can be estimated.

Our results indicate that the threshold pool mix is particularly susceptible to blending attacks since it can be rapidly flushed by an attacker, and its pool size can be accurately estimated if it is not known. The timed pool mix, and its dynamic variant, are more resistant to blending attacks since the mixing interval limits the speed with which an blending attack can be mounted. We introduced a design strategy where the designer can select the pool size and the maximum batch size so that a blending attack is unlikely to succeed before it is detected. This strategy is not possible for the threshold pool mix since the attacker can flush the mix at will.

An original designed goal of the binomial mix is to frustrate blending attacks by reducing the knowledge that the attacker has concerning the size of the pool N that must be flushed. However we have shown that the exact value of N is not required to

complete the attack with high probability. Our analysis cannot be taken further at this point without a general theory on parameter selection for the binomial mix.

There are several limitations on the analysis that has been presented, which we now discuss. Throughout the paper the attacker is assumed to be a global active attacker, who is able to monitor all communication links, delete and delay legitimate message traffic, and insert arbitrary amounts of spurious traffic. Such attackers are very powerful, designing explicit defenses against such attackers might seem unnecessary, notwithstanding the arguments given in [12] that such attackers are realistic. The analysis of the pool mixes considered in this paper is simplified by assuming such an attacker, and we may treat the results as lower bounds on the capabilities of less powerful adversaries. However it appears that the $(n-1)$-attack cannot be mounted (with certainty) by an attacker who is not globally active.

Another limitation of the analysis presented is the absence of considerations concerning traffic rates, dummy traffic, the length of mixing intervals, and a host of other practical considerations. For example we have simply assumed that an attacker can block all traffic to a mix for 20 rounds, which may correspond to an elapsed 10 hour time period in practice, and is very likely to be noticed by the mix owner. We stress however that the results of this paper are mainly aimed at the mix designer, who can then select parameters to provide security guarantees, both based on practical consideration and the analysis provided here. Previous to our work, there was no accurate basis for predicting the success of blending attacks.

References

[1] See the Maple homepage at http://www.maplesoft.com.

[2] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, July 2000.

[3] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[4] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003. Additional information on the Mixminion remailer can be found at http://mixminion.net.

[5] George Danezis and Len Sassaman. Heartbeat traffic to counter (n-1) attacks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*, Washington, DC, USA, October 2003.

[6] Claudia Díaz and Bart Preneel. Reasoning about the anonymity provided by pool mixes that generate dummy traffic. In *Proceedings of 6th Information Hiding Workshop (IH 2004)*, LNCS, Toronto, May 2004.

[7] Claudia Díaz and Bart Preneel. Taxonomy of mixes and dummy traffic. In *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, Toulouse, France, August 2004.

[8] Claudia Díaz and Andrei Serjantov. Generalising mixes. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[9] W. Feller. *An Introduction to Probability Theory and its Applications*. New York: Wiley, 3rd edition, Volume 1, 1968.

[10] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.

[11] A. Serjantov. On the anonymity of anonymity systems. Technical Report UCAM-CL-TR-604, Computer Laboratory, University of Cambridge, 2004. Available at http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-604.html.

[12] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In Fabien Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.