Generalising Mixes

Claudia Díaz[1] and Andrei Serjantov[2]

[1] K.U.Leuven ESAT-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
claudia.diaz@esat.kuleuven.ac.be
http://www.esat.kuleuven.ac.be/cosic/
&
[2] University of Cambridge Computer Laboratory
Cambridge CB3 0FD, United Kingdom
Andrei.Serjantov@cl.cam.ac.uk http://www.cl.cam.ac.uk/ aas23/

Abstract. In this paper we present a generalised framework for expressing batching strategies of a mix. First, we note that existing mixes can be represented as functions from the number of messages in the mix to the fraction of messages to be flushed.
We then show how to express existing mixes in the framework, and then suggest other mixes which arise out of that framework. We note that these cannot be expressed as pool mixes. In particular, we call *binomial mix* a timed pool mix that tosses coins and uses a probability function that depends on the number of messages inside the mix at the time of flushing. We discuss the properties of this mix.

1 Introduction

Many modern anonymity systems use the notion of a mix as introduced in [Cha81]. Chaum's original system used a very simple threshold mix, but over the last few years many different mixes have been proposed in the literature [Cot94,Jer00,KEB98].

One of the most important parameters of a mix is its *batching strategy.* Intuitively, the batching strategy of a mix is the algorithm for collecting the messages to be mixed together and forwarding them to the next hop. Naturally, this influences both the anonymity and message delay properties of the mix.

In the past the batching strategies of mixes were often described by giving the algorithm which determines when to flush the mix and how many (and which) messages to forward during the flush. In this paper, we present an simple formalism for describing mixes, which also enables a quick (qualitative) comparison. In the next section we show how existing mixes are described. In Section 4, we show that there are functions which express other mixes with interesting properties. We then focus on this mix, extend it and examine its properties.

2 Comparing Batching Strategies of Mixes

Let us examine existing mixes. There are several which we are familiar with from the literature (see survey in [SDS02]): threshold mix, timed mix, timed pool mix and the timed dynamic pool (Cottrell) mix [1]. We now seek to express mixes, just as an implementer would, as functions $P : \mathbb{N} \to [0, 1]$ from the number of messages inside the mix to the fraction of messages to be flushed. We now note that just this function is not enough to express the batching strategy of a mix. We also need to specify how often we would execute this function and flush messages. Note that in timed mixes, this is just amount to the period between mix flushes. The variable n represents the number of messages contained in the mix at the time of flushing.

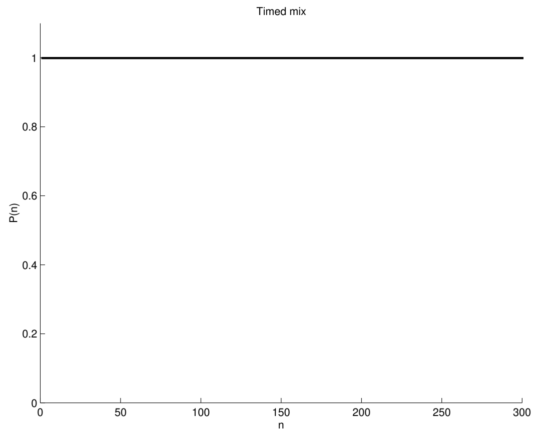
Figure 1 presents:

- Timed mix (a): This mix flushes all the messages it contains at the time of flushing. Therefore, the percentage of sent messages is always 100%, i.e., $P(n) = 1$.
- Timed pool mix (b): This mix keeps a constant number of messages, N_p, in the pool ($N_p = 20$ in this example), and flushes periodically . If the mix contains no more than N_p messages at the time of flushing, it will not output any message. When it contains more, it outputs $n - N_p$ messages, that means that the percentage of sent messages can be expressed as: $P(n) = 1 - N_p/n$.
- Timed dynamic pool mix (Cottrell mix) (c): This mix outputs messages at the timeout only when the number of messages is greater than a threshold N_p. The number of output messages is a fraction, f, of the difference between the number of messages inside the mix and the value of the threshold of the pool, $f(n - N_p)$ ($f = 0.7$ and $N_p = 20$ in the example). In the figure, the function that represents the percentage of sent messages is $P(n) = f(1 - N_p/n)$.
- Threshold pool mix (d): We have noted above that each mix is a function, together with a time period (T) which specifies how often we flush the mix. If we set $T = 0$ and let the function $P(n) = 0$ everywhere apart from the threshold, we can express threshold mixes as well as timed mixes. Thus, such a mix mixes are represented by a single dot in the figure (at $(N, 1)$ for a threshold mix, or (N,$1 - N_p/N$) for a pool mix with pool of N_p) as it is shown in Figure 1 (d). The mix shown in the figure is a threshold pool mix with threshold $N = 100$ and pool size $N_p = 50$.

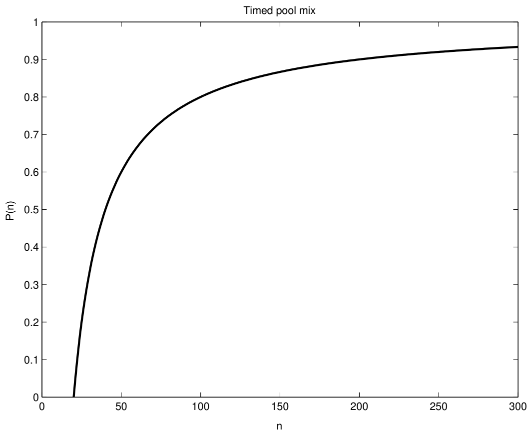
Note that the reason we have been able to express all the above mixes in this framework is that they are stateless, i.e. the fraction (and therefore the number) of messages to be flushed depends only on the number of messages in the mix, but not, say, on the number of messages flushed during the previous round.

Before proceeding to examine new $P(N)$ functions, we need to understand the effect they have on the anonymity of a mix.

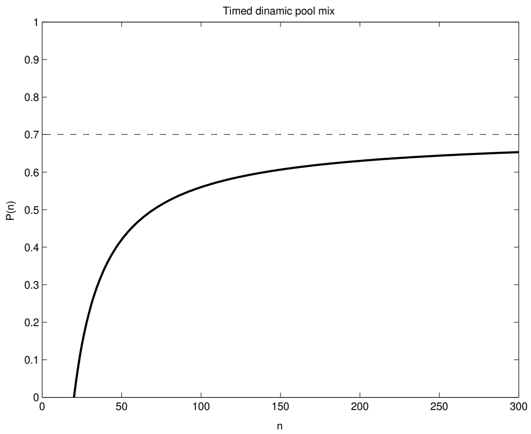
[1] The current implementation of Mixmaster uses a slightly different algorithm: it flushes a fixed fraction of the *total* number of messages in the mix, given that the number of messages that stay in the pool is larger than a minimum; otherwise, it does not send any message.



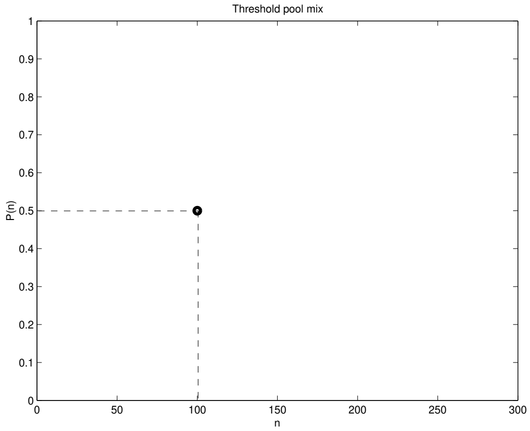
(a) Timed Mix



(b) Timed pool mix



(c) Timed dynamic pool mix



(d) Threshold pool mix

Fig. 1. Representing mixes as functions from the number of messages inside the mix to the fraction of messages to be flushed

3 Anonymity set size

We know from [SD02,DSCP02] that the anonymity set size can be computed using the entropy of the probability distribution that relates incoming and outgoing messages. This metric depends on two parameters: the number of messages mixed and the value of the distribution of probabilities of an outgoing message matching an input. In the absence of *a priori* or contextual information about the inputs, this distribution is given by the probability of a message leaving in each round. Therefore, the more messages we mix, the more anonymity; and the more evenly distributed the probability of a message leaving in round r, the more anonymity (i.e., we gain anonymity when it is more difficult to predict the number of rounds that a message stays in the pool).

Let us focus on timed pool mixes. The function represented in the Figure 1(b) gives us the probability of a message leaving in the current round as a function of the number of messages contained in the mix. Let n_r be the number of messages contained in the mix at round r, and $P(n_r)$ (the represented function) the probability of a message leaving the mix at round r.

The probability of a message that arrived at round i leaving at round r is given by:

$$prob(i) = P(n_r) \prod_{j=i}^{r-1} (1 - P(n_j)) \ .$$

That is, the fact that the message did not leave the mix in the rounds $i..(r-1)$ and it leaves in round r. Note that when $P(n_j)$ grows, the $prob(i)$ values are less evenly distributed, and the entropy (and, consequently, the anonymity set size) decreases[2]. This is not a problem if the number of messages mixed at each round is large, but when n is close to the pool size, the anonymity may be too small. We propose a solution to this problem in Section 5.

4 Generalising Mixes

The natural way to proceed is to say that a mix is an arbitrary function from the number of messages inside the mix to the percentage of messages to be flushed. What does this gain us?

Throughout the mix literature, a tradeoff between message delay and anonymity can clearly be seen. Indeed, as Serjantov and Danezis showed in [SD02], the pool mix gains more anonymity from higher average delay as compared to the threshold mix. Expressing the mix batching strategy as a function allows us to define an arbitrary tradeoff between anonymity and message delay. We now go on to examine a particular mix function.

[2] Note that this is entirely consistent with our intuition: the higher the fraction of messages we flush each round, the smaller the anonymity. Or equivalently, the more messages we delay during each round, the higher the anonymity.

5 Proposed design

Suppose that we woudl like to develop a mix which has the properties in low and high traffic conditions[3] as a particular timed dynamic pool mix, but which gains more anonymity for a longer delay in low traffic conditions. This is easily possible – all one needs to do is to invent a suitable function.

Note that the numbers are given on order to illustrate qualitative examples. The values of the functions should be optimised for the requirements of a particular system, depending on the traffic load, number of users, tolerated delay, minimum required anonymity, etc.

In Figure 2 we show a comparison between the timed dynamic pool mix and our new mix, which is defined by a suitable function (normal cumulative distribution function).

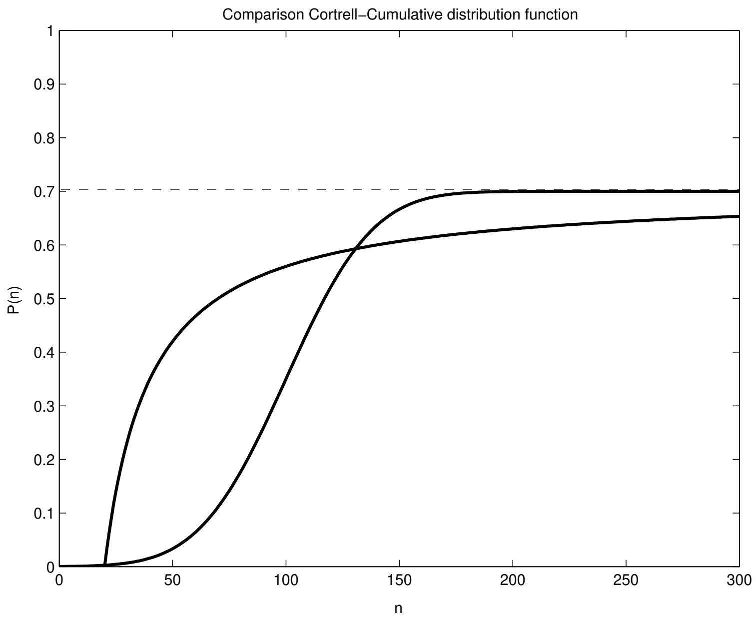


Fig. 2. Timed dynamic pool mix vs a mix based on the normal cumulative distribution function

The normal cumulative distribution function has desirable properties. It grows smoothly at low n, providing a larger anonymity when the mix contains few messages. This is achieved at the cost of increasing the delay in low traffic

[3] Or, more pragmatically, the same size of the pool and the same fraction of messages to be sent out when there is lots of traffic

conditions. On the other hand, when the number of mixed messages is large enough, the cumulative function improves the delay over the Cottrell function.

6 Randomising Mixes: The Binomial Mix

In this section we add randomness to a mix. This has the effect of hiding the number of messages which are in the mix at the time it flushes.

Suppose we treat the result of the function $P(n)$ not as a fraction, but as a probability. We can then use it as the bias of a coin, which we toss for each message inside the mix. A head indicates that this message should be sent out during this round, a tail – that it should remain in the mix.

Let s be the variable that represents the number of messages sent by the mix when it flushes. On average, $s = nP(n)$; but s follows a binomial distribution, which has a variance equal to $np(1-p)$, where p is the result of the function $P(n)$. The property of the mix is that by observing s the attacker does not obtain much information about the value of n. The effort required to estimate n is analysed in Section 6.1.

Due to this property, we call this proposed mix *binomial mix*.

6.1 Guessing the number of messages contained in the mix

We analyse the information obtained by a passive attacker that observes the input and output of the binomial mix. Then we explain how the attacker can combine the information obtained in multiple observations and give an estimate of the number of rounds needed to accurately guess n.

Observation of one output. When the attacker is allowed to observe only one output, the only available information he has is s. We have constructed a simulator that calculates the probabilities of every value of n after observing that the mix outputs s messages.

Given n, we can calculate the probability of sending s messages with the following formula, according to the binomial distribution [Fel50]:

$$p(s|n) = \frac{n!}{s!(n-s)!} p^s (1-p)^{n-s} \ , \tag{1}$$

where p is the result of the function $P(n)$.

But the attacker does not know n, he has to estimate n from the observation of s. Bayes' rule can be applied to reverse the formula and compute the probability of each n[4].

$$p(n|s) = \frac{p(s|n)}{\sum_{i=s}^{N_{max}} p(i|n)} \ . \tag{2}$$

[4] Given that the attacker does not have any *a priori* information he must assume, initially, that any possible value of n between s and N_{max} (maximum capacity of the mix) is equally probable.

The attack is implemented as follows: the attacker observes s and assumes that the n that generated this output is at least s and at most N_{max}. In order to compute the probability of n taking a particular value, say 100, we apply equation 1 using this value for n, and then substitute the result in equation 2. We also need to calculate the result of equation 1 for this n and every possible value of s.

Using this formula the attacker can obtain the probability of each value of n given than the mix has sent s messages. The practical results show that the attacker cannot guess the value of n with probability greater than 15%. We have also calculated the 95% confidence interval and found that, typically, it contains between 12 and 30 different values of n. This is due to the large value of the variance of a binomial distribution.

Number of rounds needed to estimate n with 95% confidence. We have implemented a passive attack in the simulator in order to have an estimate on the number of rounds required by the attacker to guess with probability 95% the correct value of n.

Given that every round is independent from the others, we can multiply the results of every round, taking care of shifting the terms we are multiplying as many positions as the difference between the n of the first round of attack, n_0, and the current n_r. This difference is known to the attacker because he can count the incoming and outgoing messages. The details of this algorithm can be found in Appendix A.

The attacker, according to the results of the simulations, needs typically close to 200 rounds of observation. This number could be improved by choosing a more appropriate *P(n)-function*. In terms of time, he will have to wait the number of rounds times T (timeout of the mix).

6.2 The blending attack on the binomial mix

As we have seen in the previous section, a passive attacker needs a substantial number of rounds of observation in order to accurately guess the current n. Therefore, it does not seem to be practical to deploy a blending attack using the same strategy as with classical pool mixes.

In this section we describe first the attack model, then the steps needed in order to deploy a blending attack and, finally, we analyse the results.

Attack model. The attacker we are considering controls all communication lines (global attacker). He can not only observe all incoming and outgoing messages, but also delay the messages of the users and insert messages (active attacker). The attacker does not have access to the contents of the mix, i.e., the mix is a black box for the attacker (external attacker). In order to test the effectiveness of the design, we consider a setup with only one mix. which

The flooding strategy. The goal of the attacker is to trace a particular message (the target message) that is sent by a user to the mix. The actions of the attacker can be divided into two phases: the *emptying* phase and the *flushing* phase.

The emptying phase. During this stage of the attack, the goal of the attacker is to remove all unknown messages contained in the pool, while preventing new unknown messages from going into the mix. In order to force the mix to send out as many unknown messages as possible in each round, the attacker sends to the mix N_T messages, where N_T is the minimum number of messages that guarantees that the $P(n)$ function takes its maximum value, p_{max}. If the attacker wants to empty the mix with probability $1-\epsilon$, then he will have to flood the mix for r rounds.

The formula that can be used to estimate the number of rounds needed to flush all unknown messages with probability $1-\epsilon$ is:

$$(1-(1-p_{max})^r)^n \geq 1-\epsilon \ . \tag{3}$$

Where n is the number of messages contained in the pool. If the attacker does not have any information about n he will have to assume $n = N_{max}$ (worst case scenario for the attacker).

Cost of emptying the mix. We compute the cost, C_E, of this phase of the attack taking into account the following:

– Number of messages the attacker has to send to the mix.
– Time needed to complete the operation.
– Number of messages the attacker has to delay.

Number of messages the attacker has to send to the mix. In the first round the attacker has to send N_T messages, to ensure that the function $P(n)$ takes its maximum value, p_{max}, and therefore the probability of each message leaving is maximum. In the following rounds, it is enough to send as many messages as the mix outputs. Note that if $n + N_T$ is bigger than N_{max}, then some messages will be dropped and the mix will contain N_{max} messages.

Thus, for the first round the attacker sends N_T messages, and the following rounds he sends $(N_T + n)p_{max}$ messages on average. The total number of messages sent during this process is:

$$\text{Number of messages sent} = N_T + (r-1)(N_T+n)p_{max} \ . \tag{4}$$

Time needed to complete the operation. This is a timed mix, so the attacker has to wait T units of time for each round. Therefore, the total time needed is rT time units.

Number of messages the attacker has to delay. Assuming that the users generate messages following a Poisson distribution with parameter λ, the attacker has to delay, in average, λrT messages.

The flushing phase. Once the mix has been emptied of unknown messages, the attacker sends the target message to the mix. Now, he has to keep on delaying other incoming unknown messages and also send messages to make the mix flush the target.

The number of rounds needed to flush the message is, on average, $r = \frac{1}{p_{max}}$ The cost of this phase is computed according to the previous parameters.

Number of messages the attacker has to send to the mix. Assuming that the attacker carries out this phase of the attack immediately after the emptying phase, the number of messages needed in the first round is $(N_T + n - 1)p_{max}$, and in the following ones $(N_T + n)p_{max}$. The total number of messages is:

$$p_{max}(N_T + n - 1 + (r - 1)(N_T + n)) \tag{5}$$

The other two parameters are computed in the same way as in the emptying phase, taking into account the new value of r.

Guessing the number of messages within the mix with an active attack The attacker can use the flooding strategy (emptying phase only) in order to determine the number of messages contained in the pool of the mix. This attack is much faster than the one described in Section 6.1, although it requires more effort from the attacker.

Probabilistic success. Note that, due to the probabilistic nature of the binomial mix, the attacker only succeeds with probability $1 - \epsilon$. Therefore, with probability ϵ there is at least one unknown message in the mix. In this particular case, the attacker can detect his failure if during the flushing phase more than one unknown message leaves the mix in the same round (and there is no dummy traffic policy), which happens with probability p_{max}^2 for the case of one unknown message staying during the emptying phase (the most probable case). With probability $p_{max}(1 - p_{max})$ the target message leaves the mix alone, and the attack is successful. Also with probability $p_{max}(1-p_{max})$, the other unknown message leaves the mix first, and the attacker follows a message that is not the target without noticing. Finally, with probability $(1 - p_{max})^2$, both messages stay in the pool and the situation is repeated in the next round.

6.3 Average delay of a message.

Assuming that the population of users generate messages following a Poisson distribution with mean λ messages per time unit, and given that the mix flushes messages every T time units, the average number of messages going into the mix per round is λT. Assuming that the mix outputs as many messages as it gets (that is, the $P(n)$ function and N_{max} are designed in such a way that the probability of dropping messages because of a lack of space in the mix is very small), the average number of messages sent per round is $s = \lambda T$. We know that

$s = nP(n)$, therefore, we have to find n such that $nP(n) = \lambda T$. This number can be found recursively.

Given that the average number of rounds that a message spends in the mix is $\frac{1}{P(n)}$, where n has to be computed as stated above, the average delay of a message going through the binomial mix is $\frac{T}{P(n)}$ time units.

6.4 Additional measure: Timestamps.

Additional measures, like timestamps, can be used in order to prevent the blending attack. This idea has already been proposed by Kesdogan *et al.* in [KEB98] for the Stop-and-Go (SG) mixes.

SG mixes work in a different way than pool mixes: users, after choosing the path of mixes, generate a timestamp for each mix in the path that follows an exponential distribution. The message is encrypted several times, each time with the key of one of the mixes. Once an SG mix has received and decrypted a message, it keeps it in the memory a period of time equal to the delay indicated by the user. Then, it forwards the message to the next mix.

Link Timestamps. In our design, the user cannot generate timestamps for every mix in the path, because he does not know how long the message is going to be delayed in each mix. Therefore, we propose the use of link timestamps: the user generates a timestamp for the first mix and, in each of the following hops, the mix puts the timestamp on the message once the message has been taken from the pool and is going to be sent.

When a mix receives a timestamp that is too old, it drops the message. With this policy, the attacker has limited time to delay messages: if he delays the target message too long it will be dropped, and the attacker will not have any means to disclose the recipient of the message.

Using this measure prevents the attacker from delaying the target message at his will, and the attacker does not have means to deploy a blending attack (unless he knows that the message is going to be sent by the user in advance, and can empty the mix before). Therefore, in this scenario the binomial mix provides protection against the blending attack. Furthermore, the anonymity provided by the binomial mix will not be threatened by a change in the traffic load while this change, if large enough, can affect the anonymity provided by a SG mix (since SG mixes only delay messages).

Drawbacks. The use of timestamps presents practical problems, and this is the reason why we have not included them in the basic design. The most serious problem is the synchronisation of clocks. If the different computers (both users and mixes) have a deviation in their clocks, many valid messages are dropped. All entities could be synchronised using a time server, but then the security of this time server becomes an issue.

Also, timestamps are not so effective if we are dealing with corrupted mixes: a corrupted mix can put a fake timestamp on a message and give the attacker extra time to empty the following mix in the path.

7 Conclusions

We have proposed a framework with which we can generalize classical pool mixes. This model seems to be a powerful tool that gives us a new understanding of the batching strategies implemented by existing mixes. Also, new strategies that improve existing designs arise from the framework. We have proposed a cumulative distribution function in order to have a tailored anonymity/delay tradeoff that adapts to the fluctuations in the traffic load.

We have suggested a simple and intuitive way to deal with the anonymity set size provided by a mix, in which the distribution of probabilities of the number of rounds that a message stays in the pool is a function of $P(n)$.

We have added randomness to the flushing algorithm, in order to hide the number of messages contained in the mix. We have analyzed the effort required by the attacker in order to deploy passive and active attacks. The success of these attacks becomes probabilistic in contrast with classical pool mix designs.

We suggest a timestamp strategy as countermeasure to limit the power of an active attacker. If such an strategy can be securely implemented, the $n-1$ attack becomes no longer possible.

8 Future work

Some of the topics we can identify as deserving further research are:

- The analysis of the possibilities of the framework. We have proposed the cumulative distribution function as an alternative to existing mix algorithms. Other functions with interesting properties may arise from the study of the framework.
- Thorough analysis of the properties of the proposed binomial mix. We have pointed out qualitative properties of this mix. A more in-depth analysis and tests are needed in order to have a full understanding of the design and its possibilities. A method for analysing timed mixes is proposed in [SN03], which needs to be generalised to account for the binomial mix. We would also like to study the implications of the fact that mixes hide the number of messages that are inside the pool.
- Study the properties of the proposed mix when dummy traffic policies are implemented.

Acknowledgements

Claudia Díaz is funded by a research grant of the K.U.Leuven. This work was also partially supported by the IWT STWW project on Anonymity and Privacy in Electronic Services (APES), and by the Concerted Research Action (GOA) Mefisto-2000/06 of the Flemish Government. Andrei Serjantov acknowledges the support of EPSRC grant GRN24872 Wide Area Programming and EC grant PEPITO.

A Algorithm used to combine the results of different observations of the output.

The results of two observations are independent, given that the result of the Bernoulli trials do not depend on previous rounds.

Notation:

- n_j is the number of messages contained in the mix at the j-th round of attack (being n_0 -the number of messages contained in the mix when the attack starts- the number the attacker is trying to guess).
- s_j is the number of messages sent by the mis in the j-th round of attack. This number is a function of n_i.
- f_j is the number of messages that arrive to the mix during the j-th round. We take into account f_j starting from $j = 1$.
- *shift* is the difference between n_j and n_0 ($shift = n_j - n_0$). The attacker knows this number because he observes the number of incoming and outgoing messages at each round; e.g., at round 1 $shift = n_1 - n_0 = f_1 - s_0$. This number can be either positive or negative.
- P is an array that contains the result of the algorithm in the present round, taking into account all the previous rounds. The array has $N_{max}+1$ elements. $P[i]$ contains the probability of $n_0 = i$.
- A is an array that contains the probabilities of the values of n for this round. The array has $N_{max} + 1$ elements. $A[i]$ contains the probability of $n_j = i$, where j is the number of the round.

The algorithm at the j-th round is as follows:

$shift > 0$ In this case we know that $n_j > n_0$. In order to be able to multiply the result of this round to the previous ones (which have the maximum value close to n_0), we have to shift the values of A *shift* positions to the left. This way, the estimation of n_j can be used to improve our knowledge of n_0 ($n_0 = n_j - shift$).

The values we lose at the left of the array are not important, because this corresponds to impossible values of n_j: given that $n_0 \geq 0$, this implies that $n_j \geq shift$. On the other hand, at the right side of the array, we have to introduce numbers. The solution is to propagate the value of N_{max}. This makes sense because in case $n_0 \geq N_{max} - shift$ then $n_j = N_{max}$, given that once the capacity of the mix (N_{max}) has been exceeded messages are dropped.

After shifting the values of the A array, we have to rescale them in order to have a distribution of probabilities (the sum of all values must be 1).

The code in Java is as follows:

```
if (shift > 0) {
    for (int i=0; i<=N_MAX-shift; i++)
        A[i] = A[i+shift];
    for (int i=N_MAX+1-shift; i<=N_MAX; i++)
        A[i] = A[N_MAX];

    // rescaling A
    double sum = 0.0;
    for (int i=0; i<=N_MAX; i++) sum = sum + A[i];
    for (int i=0; i<=N_MAX; i++) A[i] = A[i]/sum;
}
```

$shift < 0$ This is the case in which in the present round $n_j < n_0$. We have to shift the values of the A array to the right by *shift* positions. We lose the last *shift* values, which are, again, impossible values of n_j, because $n_0 \leq N_{max}$ implies $n_j \leq N_{max} - |shift|$. At the left of the array we have to introduce values from the positions 0 to $|shift| - 1$. In this case the value we introduce is 0: we know that $n_j \geq 0$, therefore $n_0 \geq |shift|$ (note that $n_0 = n_j + |shift|$). This implies that any value of n_0 smaller than $|shift|$ is impossible.

Again, as in the previous case, we must rescale the values of A in order to obtain the new distribution.

The code in Java is as follows:

```
if (shift < 0) {
    for (int i=N_MAX; i>=-shift; i--)
        A[i] = A[i+shift];
    for (int i=0; i<-shift; i++)
        A[i] = 0.0;
    // rescaling A
    double sum = 0.0;
    for (int i=0; i<=N_MAX; i++) sum = sum + A[i];
    for (int i=0; i<=N_MAX; i++) A[i] = A[i]/sum;
}
```

$shift = 0$ In this case $n_0 = n_j$, and we can multiply both arrays (P and A) without changing A.

Multiply P and A. After shifting and rescaling the elements of the array A, we can multiply both arrays element by element. After this multiplication we have to rescale the result and we obtain the distribution of probabilities of the value of n_0 including the j-th round.

The code in Java is:

```
// multiply probabilities
double sum = 0.0;
```

```
for (int i=0; i<=N_MAX; i++) {
    P[i] = P[i]*A[i];
    sum = sum + P[i];
}
// rescaling
for (int i=0; i<=N_MAX; i++) P[i] = P[i]/sum;
```

At this point, the array P contains the current distribution of probabilities, being $P[i]$ the probability of $n_0 = i$, and taking into account the information obtained during all the rounds of attack.

References

[Cha81] David Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cot94] L. Cottrell. Mixmaster and remailer attacks, 1994. http://www.obscura.com/ loki/remailer/remailer-essay.html.

[DSCP02] Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In Paul Syverson and Roger Dingledine, editors, *Privacy Enhancing Technologies*, volume 2482 of *LNCS*, pages 54–68, San Francisco, CA, April 2002. http://petworkshop.org/2002/program.html.

[Fel50] William Feller. *An introduction to probability theory and its applications*. Wiley, 1950.

[Jer00] Anja Jerichow. *Generalisation and Security Improvement of Mix-mediated Anonymous Communication*. PhD thesis, Technischen Universitat Dresden, 2000.

[KEB98] D. Kesdogan, J. Egner, and R. Buschkes. Stop-and-go-MIXes providing probabilistic anonymity in an open system. In *Proceedings of the International Information Hiding Workshop*, April 1998.

[SD02] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In Paul Syverson and Roger Dingledine, editors, *Privacy Enhancing Technologies*, volume 2482 of *LNCS*, pages 41–53, San Francisco, CA, April 2002. http://petworkshop.org/2002/program.html.

[SDS02] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *5th Workshop on Information Hiding*, volume 2578 of *LNCS*, October 2002.

[SN03] Andrei Serjantov and Richard E. Newman. On the anonymity of timed pool mixes. In *Workshop on Privacy and Anonymity in Networked and Distributed Systems (18th IFIP International Information Security Conference)*, Athens, Greece, May 2003.