AnNotify: A Private Notification Service

Ania M. Piotrowska
University College London, UK

Jamie Hayes
University College London, UK

Nethanel Gelernter
College of Management, Academic Studies, IL

George Danezis
University College London, UK

Amir Herzberg
Bar Ilan University, IL
University of Connecticut, US

ABSTRACT

AnNotify is a scalable service for private, timely and low-cost on-line notifications, based on anonymous communication, sharding, dummy queries, and Bloom filters. We present the design and analysis of AnNotify, as well as an evaluation of its costs. We outline the design of AnNotify and calculate the concrete advantage of an adversary observing multiple queries. We present a number of extensions, such as generic presence and broadcast notifications, and applications, including notifications for incoming messages in anonymous communications, updates to private cached web and Domain Name Service (DNS) queries.

1 INTRODUCTION

A number of on-line applications require timely notifications. Mail delivery protocols notify users when a new email can be retrieved, social networking and instant messaging applications send updates of presence, and broadcast notifications carry updates of DNS or cached web records. Traditionally, notification services provide no privacy *vis-à-vis* the notification service itself, that can observe the routing of notifications from the publisher of the event to the subscriber. The fact, that particular consumers are subscribed to a particular pubisher or larger groups of publishers can revel sensitive private information about them. Thus, the privacy preserving systems, such as anonymous communication systems [7], or private presence systems [5], rely on private notifications: an adversary should not be able to observe what events a user subscribes to. In this paper we present AnNotify, a private notification service, leveraging an anonymous communication system, based on simple cryptographic constructions. The AnNotify system is designed for efficiency. Subscribers only retrieve small parts of the event database, to which we refer to as *shards*. Simple and fast cryptographic techniques, allow AnNotify to scale well, while providing rigorous privacy guarantees.

AnNotify has numerous applications. Some only require private notification to signal availability of a service or a peer (e.g., in instant-messaging systems), or events such as alerts. Other applications, e.g., blacklists, require public notifications with multiple subscribers. Broadcast notifications may signal when a cached value changes; this is especially important for privacy-preserving storage mechanisms such as Oblivious RAM [21, 33] and PIR [8], where each access involves significant overhead. Beyond these, the broadcast notifications can improve the privacy of web and DNS caches, and significantly improve the performance of such caches when they are queried over anonymizing networks such as Tor; see [16, 25, 32].

Contributions: This paper makes the following contributions:

- We introduce AnNotify, a new private and scalable notification system, which guarantees relationship privacy at a low bandwidth and performance cost.
- We present a security definition for AnNotify which allows for some leakage, to flexibly accommodate efficient systems. We also present a rigid proof of AnNotify security, delivering an upper bound of the information leakage, which can be applied to systems which security is based on sharding.
- We present an implementation of AnNotify as a web-server, which can be scaled to millions of clients at a lower cost than alternatives such as DP5 which we also evaluate.

2 MODEL AND GOALS

AnNotify is a service connecting *notification publishers* with specific *notification subscribers* that *query* for notifications. We describe the system for a single subscriber per notification first and extend it later to broadcast to multiple subscribers.

The AnNotify system consists of multiple *shards* that are managed by a single untrusted server. Shards store information about the presence of the notifications uploaded by the publishers, which subscribers can then query from the system. AnNotify operates in epochs. Each epoch publishers, who want to notify the subscriber, connect directly to the system to upload the notifications, whereas the subscribers, in order to subscribe or query for notifications, connect with the servers through the anonymous channels, as illustrated in Figure 1. AnNotify uses anonymous channels for communications, and leverages them to increase the efficiency of private queries from a database of notifications. We consider these channels to be perfect, namely to hide all meta-data about senders and receivers of messages, and also the length of messages, as would be expected from a robust mix-network [7].

Security Goals. The AnNotify system provides a number of privacy properties:

Subscriber privacy. Third parties, including the notifier and the infrastructure, cannot tell whether a subscriber sought a notification from a particular publisher.

Epoch unlinkability. An adversary cannot tell whether queries across epochs were initiated by the same subscriber or concern the same notification.

Broadcast privacy. When multiple subscribers are authorized to receive the same notification, corrupt subscribers cannot discover that other honest subscribers are subscribed to the same notification as they are.

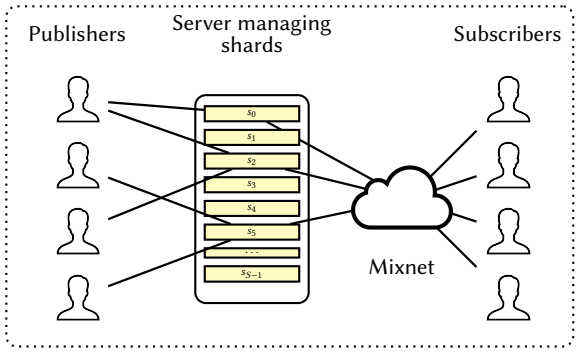


Figure 1: The AnNotify architecture.

Threat Model. The AnNotify design assumes a global passive adversary, who may observe any part or the whole network and tries to infer the relationships between publishers and subscribers. All servers that manage shards may be malicious and work with the adversary.

Moreover, AnNotify considers that a fraction of users are malicious: they collude with the eavesdropping adversary, servers or other users to try to break the privacy properties of the system or reveal some information about other users. However, we assume that a large number of concurrent AnNotify users (publishers and subscribers) are honest, and follow the protocol faithfully. We also assume, that the adversary has a partial knowledge about the relationships among publishers and subscribers , and that the adversary may chose to some extent which honest users participate in the protocols at different times. We justify those assumptions further in the paper.

All communications among the requesting subscribers and the servers go through an anonymity network [7, 10, 28]. We assume that this system is immune to traffic analysis. Namely, from the point of view of the adversary, it provides a perfect secret permutation between its input and output messages.

3 THE DESIGN OF ANNOTIFY

In this section, we present the detailed description of AnNotify. We first start with sketching the straw-man design based on trivial Private Information Retrieval (PIR), and argue informally for its security but also its inefficiency. We then present the detailed description of AnNotify.

Straw-man Design. A single server acts as the infrastructure for storing notifications. Publishers and subscribers privately agree on a secret random identifier for a specific notification event. When a publisher wishes to send a notification, she transmits the prearranged random identifier to the server which stores it forever. Subscribers of notifications access the single server, and periodically download the full database of stored notification identifiers, looking for identifiers they recognise as events. This naïve design is secure: since subscribers always download the full database, an adversary at the server cannot distinguish the notification they seek. However, performance is poor: since the database grows continuously, and downloading the full database becomes very expensive. Even using PIR [8], for more efficient private download causes a scalability bottleneck and has performance limitations, as the DP5 presence service [5] illustrates (more in Section 6.2). AnNotify provides an efficient and scalable solution to this problem, at the cost of some privacy leakage, which we evaluate carefully.

procedure $\mathcal{N}$.GenSystem(n, S, κ, Δ)
  Choose packet length l
  for $i = 0, \ldots, S-1$ **do**
    $s_i^0 \leftarrow []$
  end for
  ▷ σ is the state of the system
  $\sigma \leftarrow \{s_0^0, s_1^0, \ldots s_{S-1}^0\}$
  $\pi \leftarrow \{n, S, \kappa\}$
  return σ, l, π
end procedure

procedure $\mathcal{N}$.GenChannel(π)
  $ck \xleftarrow{R} \{0,1\}^\kappa$
  return ck
end procedure

procedure $\mathcal{N}$.Notify(ck_i, t)
  $\mu \leftarrow \mathsf{PRF}_{ck_i}(t)$
  return (i, μ)
end procedure

procedure $\mathcal{N}$.ProcNotify(μ, t, σ)
  $i \leftarrow \mu \mod \pi.S$
  if s_i^t not in σ **then**
    $s_i^t \leftarrow []$
    $\sigma' \leftarrow \sigma \cup \{s_i^t\}$
  end if
  Add string μ to Bloom filter s_i^t
  return σ'
end procedure

procedure $\mathcal{N}$.Query(ck_i, t, π)
  $\mu \leftarrow \mathsf{PRF}_{ck_i}(t)$
  $\phi \leftarrow \mu \mod \pi.S$
  $\phi' \xleftarrow{\$} \{0, \ldots, \pi.S\}$
  return $\{\phi, \phi'\}$
end procedure

procedure $\mathcal{N}$.ProcQuery(ϕ, t, σ, sk)
  $\rho \leftarrow (\phi, s_\phi^t \text{ from } \sigma)$
  return ρ
end procedure

procedure $\mathcal{N}$.ProcResponse(ρ, ck, t)
  $\mu \leftarrow \mathsf{PRF}_{ck}(t)$
  $\phi \leftarrow \mu \mod \pi.S$
  $\phi'', s \leftarrow \rho$
  if μ in s and $\phi = \phi''$ **then**
    return True
  else
    return False
  end if
end procedure

Figure 2: The concrete instantiation of all algorithms of AnNotify.

3.1 The AnNotify Protocols

Figure 2 presents the concrete algorithms of AnNotify, which we discuss informally below.

Setup. We consider a population of n users, distinguished as publishers and subscribers, using the AnNotify system to exchange notifications. We denote S as the number of *shards* used by AnNotify for sharing notifications, and each shard is denoted as $s_i, i \in \{0, \ldots, S-1\}$. To increase the capacity and scalability of the system, the shards can be distributed among multiple untrusted servers, however, the number of servers does not impact security. Thus, we consider a single untrusted server managing all shards.

AnNotify uses Bloom filters [3], an efficient data structure used for representing set membership, in order to compress the representation of the shards. We note that Bloom filters are not used in AnNotify as privacy mechanism, and could be replaced by any other (succinct or not) data representation. As $\mathcal{N}.\mathsf{GenSystem}(n, S, \kappa, \Delta)$ we denote the system setup procedure, ran by the server to initialize all parameters of the system, where $\kappa \in 1^*$, $\Delta > 0$ are the security parameters.

A publisher who wishes to send a notification to a subscriber, simply provides them with a secret *channel key* (ck) – either directly

or derived through a public key cryptographic scheme. We denote the channel establishing procedure as $\mathcal{N}$.GenChannel(π), where π is the public information.

For publishing and querying notifications clients use a cryptographic Pseudo-Random Function ($\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$) that is indistinguishable from a true random function to a computationally bound adversary not knowing the secret key [23]. The AnNotify system operates in sequential epochs, like Apres [24], denoted by t for time. For simplicity we assume that the length of all notifications, queries and responses, is always a fixed value l.

Publishing notifications. To publish a notification the publisher runs $\mathcal{N}$.Notify which derives an epoch specific notification identifier ID^t_{ck} for a particular event using a PRF. For each single notification the publisher computes the event notification identifier for epoch t using the shared channel key ck as $ID^t_{ck} = \mathsf{PRF}_{ck}(t)$. The publisher then computes the index of shard s_i in which the notification should be stored as $i \leftarrow ID^t_{ck} \bmod S$. Finally, the publisher sends ID^t_{ck} directly to the server managing shard s_i. This process spreads different notifications across shards. The server may optionally perform some authentication and authorization of publishers before accepting to store the notification. Our scheme does not impede this, but details around authenticity are outside the scope of this work.

Storing notifications. The server manages a set of shards, modeled as Bloom filters, for a given time epoch t. Upon receiving a notification ID^t_{ck} at epoch t server runs procedure $\mathcal{N}$.ProcNotify, which adds the notification to a Bloom filter $\mathsf{B}_{i,t}$ [18] for shard s_i, which includes all received notifications for a particular epoch. The server makes all shards available for download in the next epoch.

Querying for notifications. To check for notifications, subscribers repeatedly poll, in every epoch, the server for notifications by downloading the shards of interest via an anonymous network. At the beginning of epoch $t+1$ each subscriber reconstructs the epoch event identifier ID^t_{ck} for the notifications they wish to check for the previous period t by computing $ID^t_{ck} = \mathsf{PRF}_{ck}(t)$. Next, they recompute the shard identifier $i \leftarrow ID^t_{ck} \bmod S$, in which ID^t_{ck} might be stored. We denote this querying procedure as $\mathcal{N}$.Query. Alongside the query for the 'real' shard of interest each honest user anonymously sends a 'dummy' indistinguishable and unlinkable query to a random shard. These dummies ensure that no matter what side information is available to the adversary, each honest user contributes some uncertainty to the pattern of queries for the epoch. The notification service runs next the $\mathcal{N}$.ProcQuery in order to process the received queries and returns the obtained results to the subscribers. Each subscriber then *anonymously*, through a mix network, downloads the Bloom filter $\mathsf{B}_{i,t}$ for shard s_i.

Processing the reponse. Upon receiving a response from the server, the subscriber triggers the procedure $\mathcal{N}$.ProcResponse(ρ, ck, t), which checks whether ID^t_{ck} is present in the filter or not. This procedure may yield a *false positive* match, misleading the subscriber into thinking that a particular notification was present when it was not. However, selecting proper Bloom filter parameters relative to the number of notifications allows us to minimize the error probability [6].

4 SECURITY OF ANNOTIFY

In this section, we first discuss an *Indistinguishable-Notification Experiment*, a challenge game between an adversary and the system, which we use to measure the security. Next, we construe the security definition resulting from it, to quantify the privacy properties guaranteed by the notification systems. Finally, we present the main security theorem of our system and the degree of security obtained for concrete parameters.

4.1 Game between the adversary and the AnNotify system

In this section, we describe an *Indistinguishable-Notification Experiment* (IndNotExp), defined in details in Figure 7 in appendix A.1, addressing the threats identified in section 2. In this experiment, the adversary $\mathcal{A}$ observes the system over many epochs. There exists a *target subscriber*, that may be subscribed to one of two publishers (A or B) that are controlled by the adversary. The goal of the adversary is to infer to which publisher a target user is subscribed.

At the beginning of time, the experiment flips a bit b at random, and decides which of the two publishers the *target subscriber* subscribes to. Over multiple epochs the adversary schedules multiple notifications and queries to be executed, and has a full control over which honest publishers notify and which honest subscribers query for their respective notifications. We assume, that at least u honest subscribers query every epoch. $\mathcal{A}$ observes the query patterns of the subscribers, including the *target subscriber* requesting the target notifications, possibly over multiple epochs, and tries to guess b. The threat model captured by the *Indisitinguishable-Notification Experiment* is very generous to the adversary: $\mathcal{A}$ has a full visibility into the processing of all notifications and all query requests at all shards of the system for as many epochs as they wish. The adversary is also assumed to know the relationship between all honest publishers-subscriber pairs[1] and is given the secrets associated with the notifications of the two potential target notifications –modelling corrupt notifiers or other subscribers in a broadcast group. Figure 7 in Appendix illustrates the detailed IndNotExp experiment as a game in which the adversary controls, for a number of epochs, notifications ($\mathcal{A}(i, t, \texttt{'notify?'}) = 1$) and queries ($\mathcal{A}(t, u, \texttt{'GetSubscribers?'})$) from users. The adversary is given all the above information including the challenge notification keys ck_A and ck_B (through invocations to $\mathcal{A}(\cdot)$). In r rounds, the adversary may chose to trigger the target subscriber to query by setting $\mathcal{A}(t, \texttt{'TargetQuery?'})$ to 1. Finally, the adversary tries to guess a challenge bit b with $\mathcal{A}(\texttt{'Guess?'})$, i.e., tries to decide which target notification was queried by the target subscriber in the protocol run with full knowledge of the secrets it shares with notifiers. The game returns 1 if the adversary guessed correctly.

Based on the presented challenge experiment we now define a Δ–private notification system.

Definition 4.1. A notification system $\mathcal{N}$ is (u, n, Δ)-private if for any PPT adversary $\mathcal{A}$ holds:

$$\Pr\left[\text{IndNotExp}(\mathcal{N}, \mathcal{A}, n, S, \kappa, \Delta, u) = 1\right] \leq \frac{1}{2} + \Delta + \text{negl}(\kappa)$$

[1] It is inevitable to model a private notification system that leaks information. Since the adversary may observe the system for a polynomial number of past epochs she may learn all other mappings except the challenge one.

The probability is taken over all coin tosses, including uniform choice of bit b, and where negl$(\cdot)$ is a negligible function; the inequality should hold for sufficiently large security parameter κ and depends on the number of epochs r the target subscriber was activated to query. For simplicity, we call such a system Δ-private. Intuitively, Δ defines the advantage of the adversary, in successfully guessing which notification the *target subscriber repeatedly queried*, over a random guess. If the adversary would be guessing randomly, she has a 50% chances of a correct guess. Thus, Δ quantifies how much additional information the observed system leaks to $\mathcal{A}$.

This definition ensures that AnNotify provides privacy even when the adversary knows the shared key – allowing notification privacy even when the notifier or another subscriber in a broadcast group, is dishonest and working with the adversary.

4.2 The Security of AnNotify

In this section, we present the security theorem showing AnNotify to be a secure Δ-private notification system, as defined in Definition 4.1 (Section 4.1). We highlight, that the presented security theorem is very general, thus is not limited to the AnNotify system but also can be applied to other systems, which distribute the information among many entities and base there security properties on an set of honest participants. Examples of such systems are presented in section 7.

We recall, that S denotes the number of shards, u denotes the minimum number of honest subscribers querying in every epoch and r denotes the number of epochs the adversary observes the target subscriber querying for a notification.

In order to quantify the security properties of AnNotify, we want to compute the advantage of the adversary in winning the IndNotExp game, thus the chances to break the privacy of a target subscriber. We start by proving a differentially private [13] security bound ϵ for the privacy loss in IndNotExp, where the target subscriber only sends a single query.

Let us first define the following notation

Definition 4.2. Let

$$\mathcal{A} = \{(x_A, x_B) : \Pr[X_A = x_A, X_B = x_B | I_A] \leq e^{\epsilon} \Pr[X_A = x_A, X_B = x_B | I_B]\}$$

We say that $\Pr[X_A, X_B|I_A] \leq e^{\epsilon} \Pr[X_A, X_B|I_B]$ holds for $\epsilon > 0$ with probability at least $1 - \delta$ to mean that $\Pr[(X_A, X_B) \in \mathcal{A}] \geq 1 - \delta$.

In the following lemma, we quantify all the possible scenarios in which the queries sent by the subscribers are distributed among shards in such a way, that the adversary can easily link the *target subscriber* to the notification.

Lemma 4.3. *Let X_A, X_B denote the query volumes observed by the adversary at shards s_A, s_B in a single round assuming that queries map to shards following uniform multinomial distribution, and let I_A, I_B define events when a particular challenge notification is queried in the final round. An* (ϵ, δ)-differential privacy *bound by which:*

$$\Pr[X_A, X_B|I_A] \leq e^{\epsilon} \Pr[X_A, X_B|I_B]$$

holds for $\epsilon > 0$ with probability at least $1 - \delta$, where

$$\delta \leq \exp\left(-\frac{(u-1)}{4S}\right) + \exp\left(-\frac{(u-1)}{2S} \tanh^2\left(\frac{\epsilon}{2}\right)\right). \quad (1)$$

The probabilities are taken over all coin flips of honest notification not observed by the adversary.

Proof. For proof see appendix B.

Intuitively, in the presented lemma ϵ is a measure of a flexible leakage, and δ sums up the probabilities of scenarios in which the adversary is easily winning the challenge game.

Since we know how to quantify δ, we need additionally to compute the amount of leakage due to ϵ. To derive the adversary advantage for r observed queries we use a generic composition theorem. In the following lemma we derive an overall bound of adversary's advantage, in guessing to whom the target user subscribes, after r rounds when the adversary sees the target subscriber querying for the notification. As $S_{b=0}, S_{b=1}$ we denote the events that the subscriber queries a particular shard, where the target notification was uploaded. As $O_i = (X_A^i, X_B^i)$ we denote the observation of the number of queries observed coming to shard s_A and s_B respectively in round i.

Lemma 4.4. *Let O_i be an (ϵ, δ)-differentially private observation in round i, on two private inputs $S_{b=0}$ and $S_{b=1}$, for which $\Pr[O_i|S_{b=0}] \leq e^{\epsilon} \Pr[O_i|S_{b=1}]$ with probability at least $1 - \delta$.*

If the adversary $\mathcal{A}$ is provided with a set of observations over r rounds denoted as $\bar{O} = (O_1, \ldots, O_r)$ resulting from either $S_{b=0}$ or $S_{b=1}$, and tries to guess the input bit b, she succeeds with probability:

$$\Pr\left[\mathcal{A}(\bar{O}, S_{b=0}, S_{b=1}) = b \mid \bar{O}\right] \leq \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{r\epsilon}{2}\right) + r\delta + \text{negl}(\kappa),$$

where $\mathcal{A}(\bar{O}, S_{b=0}, S_{b=1})$ denotes the guess of the adversary.

Proof. For proof see appendix B.

Based on the above lemmas we derive the security theorem, proving that AnNotify is a Δ-private notification system.

Security Theorem 1. *The AnNotify system is a Δ-private notification system, for $\Delta > 0$ satisfying the following inequality. For any $\epsilon > 0$,*

$$\Delta \leq \frac{1}{2}\tanh\left(\frac{r\epsilon}{2}\right) + r\exp\left(-\frac{(u-1)}{4S}\right) + r\exp\left(-\frac{(u-1)}{2S}\tanh^2\left(\frac{\epsilon}{2}\right)\right)$$

Proof. The detailed proof is presented in Appendix A.1.

Security Theorem 1 presents a bound on Δ that provides insight about the adversary's advantage based on the security parameters of the system. The bound for Δ depends proportionaly on the ratio $\frac{u-1}{S}$ and ϵ.[2] However, this bound is very loose. A tighter bound on Δ is less elegant.

[2]Note, that the upper bound on δ in lemma 4.3 is constant as long as the ratio $\frac{u-1}{S}$ is constant. In theorem 1, because δ depends on ϵ, we obtain a uniform bound for Δ for all values of δ, when ϵ is fixed.

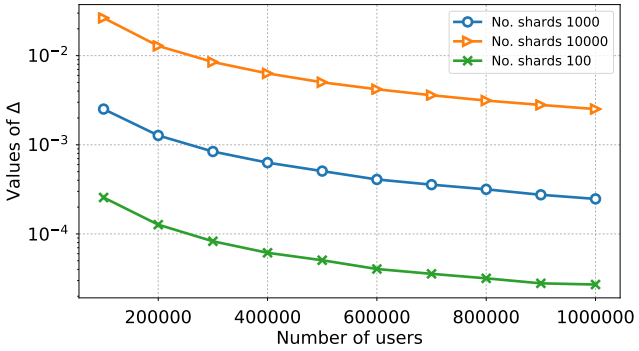


Figure 3: The empirical adversary's advantage for a single round, averaged over 10^6 samples, as a function of the number of subscribers and the number of shards. The advantage is presented on a log scale.

Lemma 4.5. *The AnNotify system is a Δ-private notification system for*

$$\Delta \leq \frac{1}{2}\tanh\left(\frac{r\epsilon}{2}\right) + r\,\mathsf{CDF}\left[u-1, \frac{2}{S}, C\right] + r\sum_{i=C}^{u-1}\mathsf{CDF}\left[i, \frac{1}{2}, \alpha\right]\binom{u-1}{i}\left(\frac{2}{S}\right)^{i}\left(1-\frac{2}{S}\right)^{u-1-i},$$

where $\epsilon > 0$.

where $\mathsf{CDF}[n, p, x]$ is the cumulative distribution function for a binomially distributed variable. We can compute this bound on Δ using Monte-Carlo integration though importance sampling.

4.3 Empirical adversary advantage

Our security theorems bound the advantage Δ of the adversary through a number of upper bounds and a generic composition theorem. This upper bound is correct but extremely loose: it assumes that in each round the worst possible observation will occur; it discounts totally cases where the adversary observes too few queries to target shards – even though they may hide information; and takes a number of loose upper bounds to yield an insightful expression. To get a more accurate view of $\hat{\Delta}$, the advantage of the adversary, we compute it empirically through sampling.

For fixed parameters u and S we draw a large number of samples from a Multinomial(θ,n) distribution with parameter vector $\theta = [(s-2)/s, 1/s, 1/s]$ and $n = u - 1$, each in effect simulating a single observed epoch. We denote as $\vec{x_A}$ and $\vec{x_B}$ the sample values falling in the second and third bucket respectively. Without loss of generality, we assume that bucket A always gets at least one message. We first compute an empirical $\hat{\delta}$ as the fraction of values in $\vec{x_B}$ that are zero, thus allowing the adversary to perfectly win the IndNotExp experiment. Given the security parameters used in the evaluation this condition is very rare and has never occurred. Next, we estimate $\hat{\epsilon}$ as the mean leakage the adversary observes for all samples with positive $\vec{x_B}$:

$$\hat{\epsilon} = \frac{1}{I}\cdot\sum_i \log\frac{x_A[i]}{x_B[i]},$$

where I denotes the number of samples. This is the log of the Geometric mean of the leakage for each epoch. From the Law of Large numbers [17], we know that for a large number of repeated experiments, the average of the results is close to the expected value, and the more trials we run, the closer the expeted value we are. Hence, the computed value of $\hat{\epsilon}$ for a large number of samples I quantifies the expected leakage of an observed round. The overall advantage after r epochs can then be computed as:

$$\hat{\Delta} = \tanh(r\hat{\epsilon}/2)/2 + r\hat{\delta}$$

This empirical advantage is the mean advantage of the adversary after observing a very large number of AnNotify epochs. And given low leakage in every round it is a more accurate depiction of the security of the system under multiple observations than the bound from our theorems. Figure 3 depicts the empirically computed adversarial advantage for a single round, over the AnNotify system composed of $10^2, 10^3, 10^4$ shards and a varying number of subscribers querying for notifications.

Further in the work, we use the empirical evaluation to accurately compare security and performance with DP5.

4.4 Other security arguments

Our main proof of security of AnNotify concerns the **subscriber privacy** property, under a very strong threat model. We argue informally in this section that other security properties also hold, but defer their formal definition and proof to a longer version of this work due to lack of space.

The **Epoch Unlinkability** property ensures that queries in different epochs cannot be linked with each other or a specific subscriber. It is a simple result of the use of keyed pseudo-random function to derive unlikable identifiers within each epoch.

The **Broadcast Privacy** property ensures that a malicious subscriber, with knowledge of the notification key, is not not able to determine whether another query (or subscriber) is querying the same known notification. This property is implied by the very strong IndNotExp definition and game. Since the adversary in this game has knowledge of the notification shared key they are exactly in the same position as another subscriber of the same notification, and thus they both enjoy at most the same advantage.

5 ANALYTICAL PERFORMANCE EVALUATION

Bandwidth. We evaluate the bandwidth cost of multi-shard AnNotify against the naïve design using a multi-server IT-PIR [8] scheme inspired by DP5 [5]. Let the number of shards in AnNotify be S, and the number of servers in the PIR scheme be S'. Since in AnNotify all shards are of equal size, denoted as l, the number of bits transferred is $nl \cdot m_x$ where n is the number of subscribers that downloaded the Bloom filter and m_x is the cost of using a mix network to transport data (to be fair we assume $m_x = S'$). For the IT-PIR scheme the cost is $nS'\sqrt{v}$, where v is the number of bits in the server's database.

Additionally, since AnNotify may yield false positives, we must consider the bandwidth cost of a subsequent action of a subscriber given that they received a notification, which we denote as a. We intentionally do not specify what this action is, as AnNotify could

be used in a variety of applications. Let $k \leq n$ be the number of subscribers who received a notification and f be the error rate of the Bloom filter. Then $h = nf$ subscribers will incorrectly think they have received a notification. Hence the cost of performing actions in AnNotify is $a(k+h)$, whereas in the PIR scheme the cost is ak since no false positives occurs.

The total cost of AnNotify is $nl \cdot m_x + a(k+h) = nl \cdot m_x + a(k+nf)$. The total cost of the PIR scheme is $nS'\sqrt{v} + ak$. We want to estimate the cutoff cost a for AnNotify to be less expensive than a PIR scheme, hence we require $nl \cdot m_x + a(k+nf) < nS'\sqrt{v} + ak$. This gives $a < \frac{S'\sqrt{v} - (l \cdot m_x)}{f}$.

We note that the false positive rate f and the size of the Bloom filter l are related by $f \approx (1/2)^{l \log 2/m}$, where m is the number of messages in the filter, that we assume is approximately N/S where N is the total number of notifications. Similarly, the database in an IT-PIR system would need at least $v = N \log N$ bits to store a list of up to N distinct notifications. Thus, it is preferable to use the AnNotify system over IT-PIR when the cost of an action a is lower than the following threshold: $a < (S'\sqrt{N \log N} - (l \cdot m_x))2^{\frac{lS}{N} \log 2}$.

Latency. In the AnNotify system, a notification sent by a publisher in epoch e_i becomes available to a subscriber in epoch e_{i+1}. The time between a notification being sent and when it can be read is $|e| + t$, where t is the round trip time taken by the notification to be routed through the mix network and $|e|$ denotes the server epoch length. Note, that this time t is dependent on the amount of traffic passing through the mix network, and the mix networks flushing mechanism.

Refresh rate, epoch length, cost and privacy.

In AnNotify system publishers and subscribers must decide on an epoch length, based on which their notification identifiers will change. There is a clear trade-off: shorter epochs mean shorter waiting times but result in the subscribers requesting more often. Publisher-subscriber epoch lengths are entirely context dependent, for example a social network presence notification system will likely have much shorter publisher-subscriber epoch lengths than a storage system.

6 EXPERIMENTAL EVALUATION

Three key advantages of AnNotify over previous works [5, 8] are efficiency, extremely low infrastructure cost (even at large scale), and ease of implementation. In this section, we describe a prototype implementation of AnNotify, based on web technologies for the server components, and Tor as an anonymity system. Next, we compare it with DP5.

6.1 Implementation & Infrastructure

We implement AnNotify as a web-server that subscribers may easily access through the most popular anonymity network today, Tor [12]. We note, that even though we use Tor, the anonymous channels might be implemented using other design, for example [28]. We are aware that Tor only provides anonymity properties against a local or limited passive adversary, and thus the experimental system inherits this limitation. Since we are concerned with performance we focus on supporting as many clients as possible, and decreasing the connection time between the client and the server.

Our implementation of AnNotify consists of two servers: a front-end server with whom the clients communicate to download shards, and a back-end server that maintains the Bloom filters. We design AnNotify so that queries are served as requests for a static resource: since those only need to retrieve the Bloom filter corresponding to a previous epoch. The task of the front-end server is simply to serve medium to large static resources; since servers are untrusted, caching and content distribution network may be used to speed this up – and this is a feature of AnNotify. We expect the size of the Bloom filter served to be similar to the size of an image, between several kilobytes to a few megabytes.

To perform a query and retrieve the Bloom filter, AnNotify clients just send an HTTP GET requests to the front-end server. To optionally register a notification, the clients can additionally send the notification identifier for the current epoch as a parameter to the HTTP request. The front-end server immediately responds with the relevant *current* Bloom filter, that is stored as a static file, and forwards the request to the back-end server to update the *next* filter. At the beginning of every epoch, the back-end server sends the *next* Bloom filters, one for each shard, to the front-end server, and the front-end server replaces the *current* Bloom filter with it.

We used Nginx[3] for the front-end server due to its high performance in serving static resources. We implemented the back-end server in Java, relying on Netty[4], a non-blocking I/O (NIO) client-server framework. We relied on Google Guava's implementation of Bloom filter[5]. The front-end implementation simply consists of the Nginx configuration file, and the back-end is 300 lines of Java code.

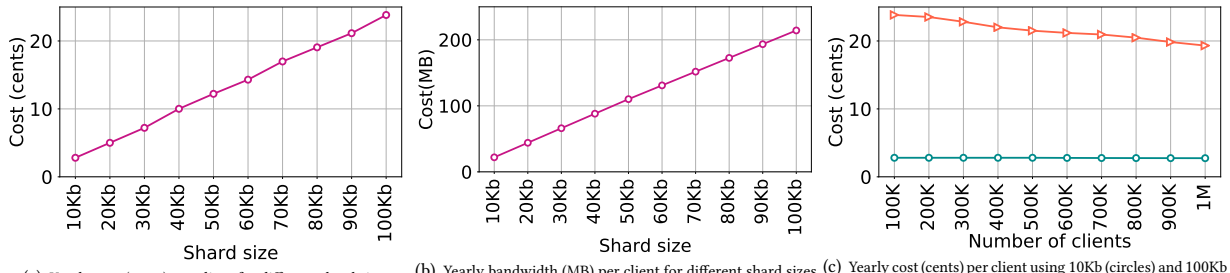
6.2 Performance Evaluation

To evaluate AnNotify, we run an AnNotify server on a single Windows 7 OS, 8GB RAM machine. The back-end and the front-end servers run as two processes. From another machine, we run our client program from several processes to simulate 100K requests in epochs of 5 minutes. We tested the system for shards from 10Kb to 100Kb. Larger shards imply larger Bloom filters to retrieve and higher bandwidth.

A single machine served 100K clients when the shard size was up to 30Kb. For larger shards we encountered sporadic failures for some clients, and had to add additional servers to handle some shards. The design of AnNotify allows distributing the shards among several machines without overhead. The yearly cost of an Amazon EC2 `m4.large` instance (in April 2016), which is equivalent to the machine we used, is $603. Dividing the cost of additional machine by 100K clients implies minimal additional cost of less than a single cent per client. Our measurements indicate an additional server is required for each 30Kb increase of the shard size.

We estimated the cost of running AnNotify in the Amazon cloud. The main factor in the cost calculation was the bandwidth that increases linearly as a function of the shard size. However, the bandwidth cost per byte decreases as the system consumes more bandwidth, e.g., for larger shards and for more clients. Figure 4

[3]The NGINX Web Server https://www.nginx.com/
[4]The Netty Framework http://netty.io/
[5]Guava: Google Core Libraries for Java https://github.com/google/guava



(a) Yearly cost (cents) per client for different shard sizes

(b) Yearly bandwidth (MB) per client for different shard sizes

(c) Yearly cost (cents) per client using 10Kb (circles) and 100Kb (triangles) shards, for different numbers of clients

Figure 4: AnNotify's implementation evaluation summary. The system scales perfectly for the increasing number of clients. Larger shards imply higher bandwidth and cost per client. The cost evaluation was done based on Amazon EC2 `m4.large` instances.

illustrates our costs estimation, extrapolated from measurements using our experimental setup, for a full year of operation in the Amazon cloud. The costs are illustrated in monetary values, on the basis of the cost of an Amazon EC2 `m4.large` instances. The results show that AnNotify is indeed very efficient, and extremely cheap to operate in the real world. Figure 4(a) shows that the yearly cost per client ranges from a few cents (shards of 10Kb) to less than a quarter (shards of 100Kb). Figure 4(b) shows the linear growth in the yearly bandwidth used by AnNotify client as a function of a shard size. However, as depicted by Figure 4(c), the AnNotify scales perfectly in the number of clients, such that the cost per client even decreases as there are more clients in the system. For a shard of size 10Kb, yearly costs per client is around 3 cents for both 100K and 1 milion users. In comparison, in DP5 the monthly cost per-user for bandwidth is about 0.05 cent, which results in 60 cents per year for 100K users, and around 120 cents for 1 milion users.

6.3 Comparison to DP5

Compared to the thousands of lines of C++ and Python used to build DP5 [5], AnNotify was significantly easier to implement and does not require PIR services or Pairing-friendly libraries. Despite being implemented in Java, it efficiently supports a hundred thousand clients, and can be parallelized to scale to millions of clients easily (see Figure 4(c)) with significantly lower yearly cost than DP5, of a few cents per client.

Given the different threat models and functionality it is delicate to provide a fair comparison between DP5 and AnNotify calibrated in terms of security. To do so we compare the second phase of DP5, with each user having a single friend, and the status communicated being a single bit notification. Thus, for u users DP5 would have to serve through PIR a database of at least u bits using IT-PIR over ℓ servers, acting as the security parameter. We configure AnNotify to also serve a database of u bits over S shards, using a mix network with path length ℓ. Both ℓ and $S < u$ are the security parameters of AnNotify for a fixed number of users u. We do not use Bloom filters to avoid making assumption on notification utilization, thus presenting a very costly variant of AnNotify.

We consider that either IT-PIR servers or mix servers may be corrupt with a fixed probability f. In that case the advantage of the adversary in DP5 is f^ℓ, namely the probability that all PIR servers are corrupt. For AnNotify the advantage of the adversary is the

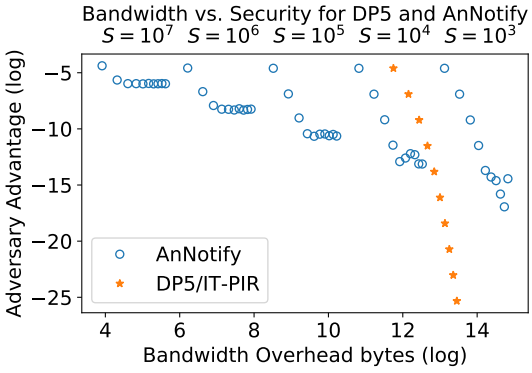


Figure 5: Security versus Bandwidth comparison for AnNotify and DP5/IT-PIR, for different parameters $\ell \in \{2, \ldots, 11\}$ and $S \in \{10^3, \ldots, 10^8\}$. Database of $u = 10^9$ bits and users, and fraction of corrupt nodes $f = 10\%$. We observe that AnNotify is orders of magnitude cheaper when some leakage may be tolerated (adversary advantage $e^{-5} \ldots e^{-10}$). (Smaller is better on both axes.)

leakage Δ, that we compute empirically (to get a tight estimate, see appendix 4.3), added to the probability f^ℓ that all mix-servers are corrupt.

Bandwidth. Figure 5 illustrates the trade-off between security and bandwidth for AnNotify compared to DP5 using the above configuration, for differing security parameters S (shards) and ℓ (mix or PIR servers). We vary $S \in \{10^3, \ldots, 10^8\}$ and $\ell \in \{2, \ldots, 11\}$. The measurements are for one billion notifications (u=10^9) and a fraction $f = 10\%$ of corrupt servers. We observe that AnNotify requires many orders of magnitude (log scale x axis) lower bandwidth per query than DP5 for moderate adversary advantage (e.g., $e^{-5} \ldots e^{-11}$). This advantage is comparable to using $\ell \leq 5$ PIR servers. For each value of S we observe that at first the advantage is dominated by the probability of the mix network failing (for low ℓ) before stabilizing and being dominated by the leakage of AnNotify.

Processing. We implement the DP5 second phase IT-PIR scheme using 64 bit *numpy* matrix multiplication, to compare the CPU costs of AnNotify versus DP5. We note that IT-PIR is CPU bound, while the untrusted servers of AnNotify are purely network bound,

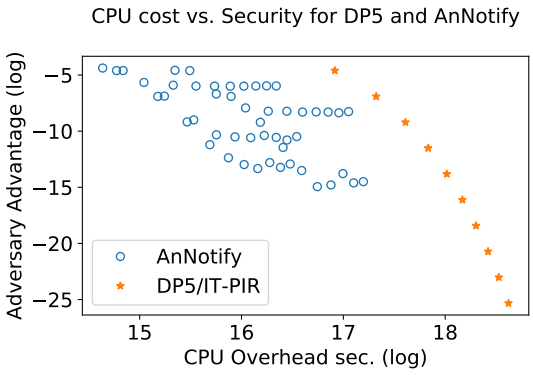


Figure 6: Security versus CPU cost comparison for AnNotify and DP5/IT-PIR, for different parameters $\ell \in \{2, \ldots, 11\}$ and $S \in \{10^3, \ldots, 10^8\}$. Database of $u = 10^9$ bits and users, and fraction of corrupt nodes $f = 10\%$. We observe that AnNotify requires less processing time than IT-PIR for comparable security levels. (Smaller is better on both axes.)

since no processing takes place on them aside from serving static shards of data. However, the anonymity network used by AnNotify may become a CPU bottleneck. To estimate this cost we measured the total CPU overhead per mix message using the Sphinx[6] packet format [10] for appropriate payload sizes and path lengths ℓ.

Figure 6 illustrates the total CPU costs for $u = 10^9$ queries and $f = 10\%$ for both DP5/IT-PIR and AnNotify. We vary $S \in \{10^3, \ldots, 10^8\}$ and $\ell \in \{2, \ldots, 11\}$. We observe that for equivalent security levels the CPU cost of mixing messages in AnNotify is always orders of magnitude (log scale x axis) lower than the equivalent CPU cost of processing IT-PIR queries in DP5.

7 ANNOTIFY EXTENSIONS

AnNotify as a presence system. AnNotify can be used as a privacy-friendly presence system, to transmit a small amount of information from the publisher to the subscriber. A presence system allows to indicate an online presence of the users. For example, when a single user connects to the network the presence system informs which friends are online.

In this variant, each shard stores the received notifications as a list within each shard, instead of Bloom filter. Two users who would like to use AnNotify share a secret channel key ck. Alice wants to notify Bob of message m on this channel. To do so, she computes the value of a pseudo random function keyed with ck based on the current time stamp as $ID^t = \mathsf{PRF}_{ck}(t)$ and the shard index $i = \mathsf{PRF}_{ck}(t) \bmod S$. She then encrypts the selected message with an Authenticated Encryption Scheme with Associated Data (AEAD) (such as AES-GCM) with a secret key ck to obtain the ciphertext $c^t = \mathsf{AEAD}_{ck}(ID^t; \mathsf{m})$. In order to notify, Alice sends the tuple (ID^t, c) to the corresponding shard s_i based on ID^t. The server adds it to the stored values within that shard.

At the beginning of the next epoch, Bob queries the servers for shard s_i and downloads the full set of values stored within it. To check for the presence notifications, the subscriber searches in the list the tuple with the identifier $\mathsf{PRF}_{ck}(t)$, and checks and decrypts the attached ciphertext and tag using secret key ck in order to recover the notification message m.

We note that the shard compression achieved through Bloom filters is sacrificed in order to transmit the message m. However, the subscriber-publisher privacy of Alice and Bob are maintained. A rigorous proof of this would have to adapt the security definition based on the IndNotExp experiment to provide the adversary with the ID^t_A and ID^t_B identifiers for the target messages instead of the raw keys ck_A and ck_B to preserve the secrecy of the message. However, the rest of the proof and Security Theorem 1 do not need major modification to show query privacy and message secrecy.

We note this scheme is in effect a leaky PIR scheme [34], based on a secure anonymity infrastructure, and untrusted servers holding shards. Given our evaluation results, relating the adversary advantage to performance, such designs may be a competitive alternative for other PIR related applications.

Broadcast AnNotify. The Security definitions and IndNotExp security game assumes that the adversary knows the notification key used by a target subscriber. Yet, they are still unable to determine whether they seek a specific notification. As a result, AnNotify can be extended to support broadcast notifications to a group, without difficulties.

In a broadcast scheme, the notifier distributes the secret notification key amongst a group of subscribers. Access control is required when publishing a notification to ensure it is genuine. This may be achieved using any authentication or non-repudiation scheme, since notifiers are not anonymous. All subscribers in the group share that key, and query each epoch on the basis of it.

Due to the security guarantees of Security Theorem 1, even if one of the subscribers in the group is corrupt – and shares the key with the adversary – they are not able to break subscriber privacy of another target user with greater advantage than the one-on-one AnNotify design.

8 APPLICATIONS

Notification-only Applications. The first application is a privacy-preserving version of event-notification services, such as the popular Yo application [36]. Yo and similar applications allow one user to send a content-free notification to peer(s). In Yo, the receiving applications notify the user by transmitting the word "Yo", in text and audio. Such event notification services can be used for social purposes, as well as to provide simple information about events, e.g., Yo was used to warn Israeli citizens of missile strikes [1].

As each message is only a single bit, applying Bloom filter is ideal for this kind of communication. The Anonymous Yo server will maintain a Bloom filter, and an anonymous Yo message will be sent by turning on a few bits according to the shared keys. The client side application will periodically retrieve the Bloom filter and will prompt Yo from another client, if this client turned on the relevant bits.

The second application is *Anonymous Presence Services*. The goal of anonymous presence services is to allow users to indicate their 'presence', i.e., availability for online communication to their peers. It is one of the functionalities usually provided by social networks

[6]Using the Python `sphinxmix` package.

such as Skype and Facebook. A privacy-preserving presence protocol, providing presence indications to users while hiding their relationships, was presented in [5]. Their solution relies on expensive cryptography and is rather complex to implement, whereas AnNotify provides an easier-to-implement and more efficient solution.

The third application is privacy-preserving *blacklists*, e.g., of phishing domain names. The goal is to allow a relying party, e.g., a browser or email server, to check if a given domain name (or other identifier) is 'blacklisted', without exposing the identity of the domain being queried. In particular, all major browsers use some 'safe browsing' blacklist to protect users from phishing and malware websites. Google Safe Browsing (GSB) alone accounts for a billion users to date [22]. To protect users privacy, clients do not lookup the suspect URL or domain-name, instead the query is for a cryptographic hash of the domain-name or URL. However, as already observed [19], providers can still identify the query. AnNotify provides an alternative which strongly protects privacy, and with comparable overhead. We note that Bloom filters are already widely used to improve efficiency of blacklists, e.g., see [18, 27].

In all applications, AnNotify allows preserving the privacy of users, by hiding the relationships between users and the notifications they receive. The use of AnNotify is easy, and has insignificant performance overhead in addition to the use of anonymous channels. However, notice that AnNotify exposes the *total* number of clients currently connected to the system. We believe this is not a concern in many applications. Indeed, many services publish an estimate of the number of online clients, e.g., see Tor metrics [29].

Privacy-Preserving Caching and Storage Services. A classical use for Bloom filters, is to improve the efficiency of caching and storage mechanisms, by allowing efficient detection when cached items were updated (or not). In particular, Bloom filters were used to improve the efficiency of web-caches [6, 15].

AnNotify can similarly improve the efficiency of caching and storage mechanisms, while also protecting privacy. This is especially important for privacy-preserving storage mechanisms such as Oblivious RAM [21, 33] and PIR [8], where each access involves significant overhead, hence avoiding unnecessary requests has a large impact on performance.

Due to its high efficiency, AnNotify can also be used to improve the privacy of web and DNS caches. In particular, web-users may use AnNotify to improve the efficiency of anonymous-browsing mechanisms such as Tor [29] and the use of AnNotify seems to offer significant performance improvements compared to existing proposals for protecting privacy of DNS users, see [16, 25, 32].

9 RELATED WORK

Bloom Filters. Extensions of Bloom filters support additional features, like deletion [4, 15, 30] or representing multisets [9]. In [2] authors presented metrics as K-anonymity and $\gamma-$deniability to measure the privacy and utility of Bloom filters but the resulting privacy properties are weak. RAPPOR [14] allows the private collection of crowd sourced statistics as randomized responses in Bloom filters, while guaranteeing ϵ-differential privacy. RAPPOR uses input perturbation locally on the client side, however extracting results requires sophisticated statistical techniques.

Anonymity. The most widely deployed anonymity system is Tor [12]. In Tor, communications are routed through a network of relays using onion routing, which hides the senders location and ensures unlinkability between the user and the visited website. Although Tor is popular it is vulnerable to traffic analysis attacks, and for stronger anonymity properties mix networks have to be used [7, 10] and as recent research showed, without sacrificing the latency [28]. Receiver anonymity systems, such as nymservers [26], may also be used to route notifications to users. Pynchon Gate [31] proposes a pseudonymous message retrieval system based on a distributed PIR scheme.

Privacy in Remote Storage. Private information retrieval (PIR) allows a client to retrieve privately a single record from a remote public database. The naive solution retrieves all records from the database, but PIR protocols are more efficient in terms of bandwidth [8, 11, 20]. IT-PIR is a multiple server PIR variant, where each server stores a replicated copy of the database. IT-PIR guarantees perfect privacy, as long as one server is honest, but requires all servers to process each query and operate on the whole database, which increases both the computational and communication costs.

Toledo et al. [34] present variants of IT-PIR based schemes composed with an anonymity systems, which reduce the computational costs by allowing some information leakage. The key difference between [34] and this work, is that AnNotify servers are entirely untrusted and it wholly relies on an anonymity system for privacy.

Social applications require private presence notifications. Traditional implementations of presence give a central server the social graph of users. Protocols like Apres [24] and DP5 [5] offer privacy-preserving notification services. Apres splits the time into epochs and hides the correlation between the connectivity of the clients in every two epochs. DP5 offers stronger privacy guarantees, however this design uses multi-server IT-PIR to look up other users presence without revealing information about the social graph. We compare this work with DP5 in our evaluation section.

The anonymous messaging system presented in Vuvuzela [35] also introduces an auxiliary scheme for notifying users, that someone wants to contact them, by sending invitations. AnNotify has a lower bandwidth and operational cost than Vuvuzela, since in that scheme the users have to download and try to decrypt all the invitations, including cover ones.

10 CONCLUSIONS

AnNotify provides efficient and private notifications in a scalable manner, compared with previous approaches like DP5 [5] that struggles to scale past 1 million users. AnNotify benefits from a mass of users: its key security parameters depend on the number of shards and anonymity set size of the underlying anonymity system. These may be tuned to provide meaningful privacy protection despite some leakage.

AnNotify lowers the quality of protection to achieve scalability, but does so in a controlled and well understood manner: the concrete security theorems presented indicate the advantage of the adversary. The tighter bounds and empirical estimates of leakage

under repeated queries provide even stronger evidence that AnNotify can provide strong protections. This is particularly relevant for large-scale deployments and applications requiring notifications, that today benefit from no protections at all.

Besides securing notifications, the AnNotify design, provides a couple of important insights into general privacy engineering. We show that anonymous channels may be important building blocks to implement schemes, such as notifications, that are not entirely related to messaging per se. Their study should expand to provide robust and efficient schemes for such applications.

PIR schemes inspired from the AnNotify design and anonymous channels, may be more competitive in terms of performance that those proposed so far, despite leakage and required large anonymity set. Pursuing this research direction would allow wider deployment of private querying in general.

11 ACKNOWLEDGEMENTS

Acknowledgements. The authors would like to acknowledge their financial support: George Danezis and Ania Piotrowska are supported in part by EPSRC Grant EP/M013286/1 and H2020 Grant PANORAMIX (ref.\653497). Jamie Hayes is supported by the UK Government Communications Headquarters (GCHQ), as part of University College London's status as a recognised Academic Centre of Excellence in Cyber Security Research. Amir Herzberg is supported by EPSRC Grant EP/M013286/1.

REFERENCES

[1] BBC. 2014. "Yo app warns Israeli citizens of missile strikes". Online. (July 2014).
[2] G. Bianchi, L. Bracciale, and P. Loreti. 2012. Better Than Nothing. Privacy with Bloom Filters: To What Extent?. In *Privacy in Statistical Databases - PSD 2012, Palermo, Italy, September 26-28, 2012*. 348–363. http://dx.doi.org/10.1007/978-3-642-33627-0
[3] B. Bloom. 1970. Space/Time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (July 1970), 422–426.
[4] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. 2006. An Improved Construction for Counting Bloom Filters. In *Algorithms âĂŞ ESA*. UK. https://doi.org/10.1007/11841036_61
[5] N. Borisov, G. Danezis, and I. Goldberg. 2015. DP5: A Private Presence Service. *PoPETs* 2015, 2 (2015), 4–24. http://www.degruyter.com/view/j/popets.2015.2015.issue-2/popets-2015-0008/popets-2015-0008.xml
[6] A. Broder and M. Mitzenmacher. 2004. Network applications of Bloom filters: A survey. *Internet mathematics* 1, 4 (2004), 485–509.
[7] D. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (1981), 84–88. https://doi.org/10.1145/358549.358563
[8] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. 1998. Private information retrieval. *Journal of the ACM (JACM)* 45, 6 (1998), 965–981.
[9] S. Cohen and Y. Matias. 2003. Spectral Bloom Filters. In *Conference on Management of Data (SIGMOD) (SIGMOD '03)*. ACM, New York, NY, USA, 241–252. https://doi.org/10.1145/872757.872787
[10] George Danezis and Ian Goldberg. 2009. Sphinx: A Compact and Provably Secure Mix Format. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. 269–282.
[11] C. Devet, I. Goldberg, and N. Heninger. 2012. Optimally robust private information retrieval. In *USENIX Security 12*. 269–283.
[12] R. Dingledine, N. Mathewson, and P. F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security*. USENIX, 303–320. http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html
[13] C. Dwork. 2006. Differential Privacy. *ICALP* (2006).
[14] Ú. Erlingsson, V. Pihur, and A. Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *CCS*. ACM, USA, 1054–1067. https://doi.org/10.1145/2660267.2660348
[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. 2000. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. *IEEE/ACM Trans. Netw.* 8, 3 (June 2000), 281–293. https://doi.org/10.1109/90.851975
[16] H. Federrath, K.P. Fuchs, D. Herrmann, and C. Piosecny. 2011. Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-Based Protection Methods. In *Computer Security - ESORICS 2011, Leuven, Belgium, September 12-14, 2011*. http://dx.doi.org/10.1007/978-3-642-23822-2
[17] William Feller. 1968. *An introduction to probability theory and its applications: volume I*. Vol. 3. John Wiley & Sons New York.
[18] S. Geravand and M. Ahmadi. 2013. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks* 57, 18 (2013), 4047–4064. http://dx.doi.org/10.1016/j.comnet.2013.09.003
[19] T. Gerbet, A. Kumar, and C. Lauradoux. 2015. *A Privacy Analysis of Google and Yandex Safe Browsing*. Technical Report Research Report RR-8686. INRIA. https://hal.inria.fr/hal-01120186v4
[20] I. Goldberg. 2007. Improving the robustness of private information retrieval. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 131–148.
[21] O. Goldreich and R. Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.
[22] Google. 2014. Google Transparency Report - Making the web safer. (June 2014).
[23] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to modern cryptography*. CRC press.
[24] B. Laurie. 2004. Apres-a system for anonymous presence. (2004).
[25] Y. Lu and G. Tsudik. 2010. Towards Plugging Privacy Leaks in the Domain Name System. In *Peer-to-Peer Computing*. IEEE, 1–10. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5569876
[26] D. Mazières and M. F. Kaashoek. 1998. The Design, Implementation and Operation of an Email Pseudonym Server. In *Conference on Computer and Communications Security (CCS '98)*. 27–36. https://doi.org/10.1145/288090.288098
[27] S. Di Paola and D. Lombardo. 2011. Protecting against DNS Reflection Attacks with Bloom Filters. In *DIMVA*, Thorsten Holz and Herbert Bos (Eds.). http://dx.doi.org/10.1007/978-3-642-22424-9
[28] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. *USENIX Security Symposium* (2017). https://github.com/UCL-InfoSec/loopix
[29] The Tor project. 2016. Tor Metrics. https://metrics.torproject.org/. (April 2016).
[30] C. E. Rothenberg, C. Macapuna, F. L. Verdi, and M. F. Magalhães. 2010. The Deletable Bloom filter: A new member of the Bloom family. *CoRR* abs/1005.0352 (2010). http://arxiv.org/abs/1005.0352
[31] L. Sassaman, B. Cohen, and N. Mathewson. 2005. The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society (WPES '05)*. 1–9. https://doi.org/10.1145/1102199.1102201
[32] H. Shulman. 2015. Pretty Bad Privacy: Pitfalls of DNS Encryption. In *Workshop on Privacy in the Electronic Society, WPES 2014*.
[33] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. 2013. Path oram: An extremely simple oblivious ram protocol. In *Computer & communications security (ACM CCS)*. ACM, 299–310.
[34] Raphael R Toledo, George Danezis, and Ian Goldberg. 2016. Lower-Cost ϵ-Private Information Retrieval. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 184–201.
[35] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 137–152.
[36] Wikipedia. 2016. Yo (app). https://en.wikipedia.org/wiki/Yo_(app). (April 2016).

A PROOFS

A.1 Proof of Security Theorem 1

Proof. To prove the main security theorem, and ultimately show that AnNotify is Δ-private, we need to show that the adversary can only win the *Indisitinguishable-Notification* game, showed in Figure 7 with an advantage Δ, defined in Definition 4.1, over a random guess. We do so by first arguing that the adversary learns nothing new[7] from rounds not including the target subscriber, and then computing the advantage given the information about the rounds when the target subscriber was active.

We proceed through a sequence of hybrid games, with slight modifications over the initial security Definition 4.1, including the IndNotExp experiment (Game_0). We first note that in the concrete protocols $\mathcal{N}$.Notify and $\mathcal{N}$.Query act on notification IDs generated using a pseudo-random function (PRF) keyed with an unknown key to the adversary and the epoch number ($ID^t = \text{PRF}_{ck}(t)$). Thus,

[7] Remember that the adversary already is assumed to know the correspondence between honest subscriber-publisher pairs, besides the target query in the challenge round.

```
procedure IndNotExp(N, A, n, S, κ, Δ, u)
    (σ, l, π) ← N.GenSystem(n, S, κ, Δ).
    ▷ Generate two challenge Publishers.
    ck_A ← N.GenChannel(π)
    ck_B ← N.GenChannel(π)
    A(ck_A, ck_B, n, S, κ, Δ, π)
    b ←$ {0, 1}
    ck_T ← (if b = 0 then ck_A else ck_B)
    ▷ Generate all other Publishers & Subscribers.
    for i = 0, . . . , n do
        ck_i ← N.GenChannel(π)
    end for
    ▷ Perform many rounds of the protocols.
    for t = 0, . . . do
        Ψ_t, Φ_t ← {}, {}
        ▷ Trigger some Publishers.
        for i ∈ {0, . . . , n} ∪ {A, B} do
            ▷ Adv. chooses notifications.
            if A(i, t, 'notify?') = 1 then
                μ_i ← N.Notify(ck_i, t)
                σ ← N.ProcNotify(μ_i, t, σ)
                Ψ_t ← Ψ_t ∪ {(i, μ_i, σ)}
            end if
        end for
        ▷ Adv. sees all notifications and server state.
        A(t, Ψ_t)
        ▷ Trigger at least u honest Subscribers.
        Q_t ← {}
        U_t ← A(t, u, 'GetSubscribers?')
        U_t ← U_t ∩ {0, . . . , n}
        if |U_t| < u then
            return 0
        end if
        ▷ Challenge the target Subscriber.
        if A(t, 'TargetQuery?') = 1 then
            U_t ← U_t ∩ {T}
        end if
        for all j ∈ U_t do
            Q_t ← Q_t ∪ N.Query(ck_j, t, π)
        end for
        for all φ_j ∈ Q_t do
            ρ_j, σ ← N.ProcQuery(φ_j, t, σ)
            Φ_t ← Φ_t ∪ {(φ_j, ρ_j, σ)}
        end for
        ▷ Adv. sees all queries and server state.
        A(t, Φ_t)
    end for
    return A('Guess?') = b
end procedure
```

Figure 7: The Indistinguishable-Notification Experiment.

from the adversaries point of view, the *ID*s and the shards selection look random and the adversary cannot learn the notification or shard number of any other entity. Hence, we can replace all instances of the first invocation of the PRF by true random functions (Game_1). Thus, the adversary can only distinguish between the original experiment Game_0 and Game_1 with negligible advantage due to the properties of secure PRFs.

In Game_1 the information within each epoch not including the target subscriber is statistically independent from the challenge b. Based on this observation, we define Game_2, that consists only of rounds in which the target subscriber is activated to query. Thus, the advantage of the adversary winning Game_2 is equal to winning Game_1.

In each of the remaining rounds of Game_2 the security definition dictates that a number u of honest users (including the target subscriber), query for their sought notification and a dummy shard.

In Game_2 the adversary can observe the ID^t for all notifications that have been seen in each epoch. However there remain u' queries ($u \leq u' \leq 2u$) for which the adversary does not know the corresponding ID^t. These are indistinguishable from a random string, and the corresponding queries are distributed uniformly among the shards S. Thus, we define Game_3 in which we simply remove all notifications and queries for which the adversary knows the ID^t from all epochs – and that does not increase the adversary advantage.

Following this, Game_3 consists of epochs within which the uncertainty of the adversary is whether notification A or notification B was queried (depending on the challenge bit b), and the volumes of at least u randomly distributed queries across all shards. Thus, for every epoch, the adversary knowing the secret keys ck_A, ck_B now has to decide on the basis of the query volumes X_A and X_B observed in the shard s_A, s_B corresponding to μ_A and μ_B respectively, what the challenge b was.

We compute the adversary advantage in Game_3 directly. We denote as S_A, S_B the events that the target user queried shards s_A, s_B corresponding to notifications A, B. Lemma 4.3 then shows that in a single epoch given two known shards and $u-1$ queries to uniformly random shards we can find ϵ, δ such that for notifications A and B and all query volumes observed by the adversary: $\Pr[X_A, X_B | I_A] \leq e^{\epsilon} \Pr[X_A, X_B | I_B]$ with probability at least $1 - \delta$. Lemma 4.4 then concludes the proof by showing this differentially private property can be translated to a concrete adversary advantage Δ gained by observing many epochs. □

B PROOFS OF LEMMAS

B.1 Lemma 4.3 from Section 4.2

The proof is presented in the full version of the paper https://eprint.iacr.org/2016/466

B.2 Lemma 4.4 from Section 4.2

The proof is presented in the full version of the paper https://eprint.iacr.org/2016/466