Ania Piotrowska*, Jamie Hayes, Nethanel Gelernter, George Danezis, and Amir Herzberg

AnNotify: A Private Notification Service

Abstract: AnNotify is a scalable service for private, timely and low-cost on-line notifications, based on mix-networks, sharding, dummy queries, and Bloom filters. We present the design and analysis of AnNotify, as well as an evaluation of its costs. The security of AnNotify is proved formally by first providing generic definitions and security game for a private notification system. Then we outline the design of AnNotify and calculate the concrete advantage of the 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, such as mail delivery protocols , which notify users when a new email can be retrieved. Other applications include updates of presence associated with social networking or instant messaging applications and caches. 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. However, 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. Existing privacy friendly systems which guarantee relationship privacy often rely on mechanisms such as PIR, that do not scale to large numbers of users.

AnNotify provides a private notification service, leveraging an anonymous communication system, based on simple cryptographic constructions. In brief, publishers and subscribers of notifications share cryptographic keys, that allow them to generate ever changing and unpredictable identifiers. Subscribers can poll privately for notifications using an anonymity system. AnNotify is designed for efficiency. Subscribers only retrieve small parts of the event database, which we refer to as *shards*. We use Bloom filters to compress the size of each shard. 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 [20, 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: We make the following contributions:

- We define *private notification schemes* and their security definitions which allow for perfect privacy or some leakage, to flexibly accommodate efficient systems.
- We introduce AnNotify, a new private and scalable notification system, which guarantees relationship privacy at a low bandwidth and performance cost.
- We provide a security analysis of AnNotify, showing it conforms to the definitions presented, and provide bounds for its leakage.
- We present an implementation of AnNotify as a web-server, which can be scaled to millions of clients at a lower cost than alternatives.
- We present the extensions of AnNotify which can support presence, status updates and broadcast of notifications, and generic PIR.

Outline: We start by presenting the high-level design of AnNotify in Section 2 and describe the system in detail in Section 3. In Section 4, we define the generic private notification systems and formulate their security

***Corresponding Author: Ania 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

definition, which we then use to define the key security theorems of AnNotify presented in Section 5. In Section 6 and Section 7, we discuss the costs of AnNotify and compare it to PIR / DP5 [5]. We then discuss possible extensions in Section 8, applications in Section 9 and survey the related work in Section 10. Finally, we conclude in Section 11.

2 Model and Goals

In this section, we provide a high level overview of the AnNotify system and present its threat model and the security goals.

2.1 High-level Overview of AnNotify

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

The AnNotify protocols make use of a number of infrastructure servers. Each server manages one or more *shards* that are distributed among multiple untrusted servers for better scalability. We denote shards as $s_i, i \in \{0, \ldots, S-1\}$ and their total number as S. Shards store information about the presence of the notifications uploaded by the publishers, which subscribers can then query from the system. Each shard represents the received notifications succinctly to record elements' presence as a Bloom filter of length l, using k independent hash functions, which act as random oracles. The capacity of the system increases as we add more servers to the infrastructure, however, the number of servers does not impact security.

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].

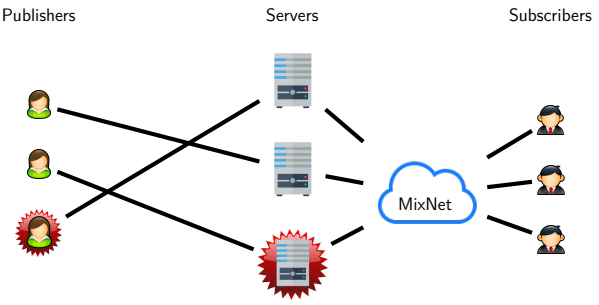


Fig. 1. The AnNotify architecture.

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.

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. Furthermore, the notifiers for a target subscriber may also 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 detail those assumptions further in the paper.

All communications among the requesting subscribers and the servers go through an anonymity network. 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

AnNotify protects the privacy of subscribers that request notifications, from a hostile network and infrastructure. We use the term *notification* for a binary signal on a predefined channel between a publisher and a subscriber, sharing a secret key, exchanged in order to indicate an event occurrence. In this section, we present the detailed description of AnNotify. We first start with sketching the straw-man design, and we argue informally for its security but also its inefficiency. We then present the detailed description of AnNotify.

Straw-man Design. The presented straw-man design is loosely inspired by trivial PIR. In this design, a single server acts as the infrastructure. Publishers and subscribers of notifications privately agree on a secret random identifier for a specific notification event. When a publisher wishes to send a notification, she simply transmits the pre-arranged random identifier to the server which stores it forever. Meanwhile, subscribers of notifications access the single server, and periodically download the full database of stored notification identifiers, looking for identifiers they recognize as events.

This naïve design is secure: since subscribers always download the full database, an adversary at the server cannot distinguish which notification they seek. However, performance is very poor, since the database grows continuously, and downloading the full database becomes very expensive. One option is to use Private Information Retrieval (PIR) [8], for more efficient private download. However, as DP5 [5] illustrates, using PIR still causes a scalability bottleneck and performance limitations. We discuss this more in Section 7.3. AnNotify provides an efficient solution to this problem, at the cost of some privacy leakage, which we carefully evaluate.

3.1 The AnNotify Design

Preliminaries. AnNotify uses Bloom filters [3] as a way of compressing the representation of the notification database. Bloom filters are used for representing a set membership of elements with some tunable *false positive* probability. The *false positive* erroneously indicates that the Bloom filter contains the queried element. A Bloom filter is defined as a bit array and constructed using several independent hash functions, which map an *inserted* element to specific filter bits. To test if a particular element was added to the set, one should compute values of the hash functions for this element and check if all corresponding bits in the filter are set. A positive answer of the check procedure may yield a *false positive* error. 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.

Setup. We consider a population of n users, distinguished as publishers and subscribers, using the AnNotify system to exchange notifications. We also assume a number of infrastructure servers, and a public mapping between servers and S shards they manage. 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. 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.

Publishing notifications. To publish a notification the publisher 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, and possibly across servers. Servers 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. Each server manages a set of shards, modeled as Bloom filters, for a given time epoch t. Upon receiving ID^t_{ck} a server adds it to a Bloom filter $\mathsf{B}_{i,t}$ for shard s_i at epoch t, which includes all received notifications for a particular epoch. To add a single identifier to the filter, a server computes the hash values of ID^t_{ck} using k, publicly available, independent hash func-

tions and sets on the resulting bits. 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 servers for notifications by downloading the shards of interest via a mix 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. Each subscriber then *anonymously*, through a mix network, downloads the Bloom filter $\mathsf{B}_{i,t}$ for shard s_i and 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 filter parameters relative to the number of notifications allows us to minimize the error probability [6]. Alongside the query for the 'real' shard of interest each honest user anonymously sends a 'dummy' indistinguishable and unlinkable query to a random shard.

We will show that AnNotify provides privacy even when the adversary knows the shared key – allowing notification privacy even when the notifier or another subscriber in a boradcast group, is dishonest and working with the adversary.

4 Private Notification Systems

In this section, we first introduce the specification of a generic private notification system. Next, we present an *Indistinguishable-Notification Experiment* and the security definition resulting from it, to quantify the privacy properties guaranteed by the notification systems.

4.1 Generic Notification System Interface

A private notification system $\mathcal{N}$ composes of a set of seven probabilistic algorithms: $\mathcal{N}.\mathsf{GenSystem}$, $\mathcal{N}.\mathsf{GenChannel}$, $\mathcal{N}.\mathsf{Notify}$, $\mathcal{N}.\mathsf{ProcNotify}$, $\mathcal{N}.\mathsf{Query}$, $\mathcal{N}.\mathsf{ProcQuery}$, and $\mathcal{N}.\mathsf{ProcResponse}$.

- $\mathcal{N}.\mathsf{GenSystem}(n, S, \kappa, \Delta)$ takes as input $n \in \mathbb{N}$ users, $S \in \mathbb{N}$ shards, and security parameters: $\kappa \in 1^*$, $\Delta > 0$ and outputs server initial state σ, packet length l and public information π relating to network and other parameters of the system necessary for publishers and subscribers to use it. This setup is ran jointly by the servers, or other notification service provider, to initialize all parameters of the system.
- $\mathcal{N}.\mathsf{GenChannel}(\pi)$ takes as input public information π and outputs channel key $ck \in \{0,1\}^*$. It is ran by notification publishers, and the resulting channel key is provided confidentially to subscribers.
- $\mathcal{N}.\mathsf{Notify}(ck, t)$ takes as input channel key ck, epoch $t \in \mathbb{N}$ and outputs a notification μ. Publishers trigger it to notify, and its output is sent to the notification service.
- $\mathcal{N}.\mathsf{ProcNotify}(\mu, t, \sigma)$ takes as input notification μ, epoch t, state σ, and outputs new state σ'. The notification service runs $\mathcal{N}.\mathsf{ProcNotify}$ after receiving each notification.
- $\mathcal{N}.\mathsf{Query}(ck, t, \pi)$ takes as input channel key ck, epoch t and a public information π, and outputs a queries $\vec{\phi}$. It is ran by the subscriber to prepare queries for a notification and the results are sent to the notification service.
- $\mathcal{N}.\mathsf{ProcQuery}(\phi, t, \sigma)$ takes as input query ϕ, epoch t, state σ, and outputs response ρ and a new state σ'. This procedure is ran by the notification service upon receiving each query, and the result is returned to the subscriber.
- $\mathcal{N}.\mathsf{ProcResponse}(\rho, ck, t)$ takes as input response ρ, channel key ck, epoch t and outputs return code ψ. This procedure is ran by a subscriber upon receiving the response from the notification service.

For simplicity we assume that the length of all notifications, queries and responses, is always a fixed value l, generated as part of the system parameters (by the $\mathcal{N}.\mathsf{GenSystem}$ algorithm).

4.2 Security Definition

In this subsection, we define an *Indistinguishable-Notification Experiment* (IndNotExp), addressing the threats identified in 2.1. In this experiment we consider an AnNotify system consisting of S shards, were n users exchange notifications. Without loss of generality we consider one server managing all shards.

In the IndNotExp experiment the adversary $\mathcal{A}$ observes the system over many epochs. There exists a special target subscriber, that may be subscribed to one of

two publishers (A or B) that are controlled by the adversary. The experiment flips a bit b at random, at the beginning of time, and decides which of these two 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 (as long as at least u honest subscribers query every epoch). She observes the query patterns of the subscribers, including the target user requesting the target notifications, possibly over multiple epochs, and tries to guess b.

Figure 2 illustrates the IndNotExp experiment as a game in which the adversary controls, for a number of epochs, notifications (through $\mathcal{A}(i, t, \text{`notify?'}) = 1$) and queries (through $\mathcal{A}(t, u, \text{`GetSubscribers?'})$) 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 $\mathcal{A}(t, \text{`TargetQuery?'})$ to 1. Finally, the adversary's goal is to guess a challenge bit $b$ with $\mathcal{A}(\text{`Guess?'})$.

Based on the presented challenge experiment we now define a $\Delta-$private notification system.

Definition 1. *A notification system $\mathcal{N}$ is (u, n, Δ)-private if for any PPT adversary $\mathcal{A}$ holds:*

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

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 may depend on the number of epochs r the target subscriber was activated to query. For simplicity, we call such a system Δ-private.*

The threat model captured by the *Indisitinguishable-Notification Experiment* is very generous to the adversary: they have has 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]. Furthermore, the adversary is given the secrets associated with the notifications of the two potential target notifications –modelling corrupt notifiers or other subscribers in a broadcast group. Ultimately, the adversary needs to decide which target notification was queried by the target

procedure IndNotExp($\mathcal{N}, \mathcal{A}, n, S, \kappa, \Delta, u$)
  $(\sigma, l, \pi) \leftarrow \mathcal{N}.\mathsf{GenSystem}(n, S, \kappa, \Delta)$.
  ▷ Generate two challenge Publishers.
  $ck_A \leftarrow \mathcal{N}.\mathsf{GenChannel}(\pi)$
  $ck_B \leftarrow \mathcal{N}.\mathsf{GenChannel}(\pi)$
  $\mathcal{A}(ck_A, ck_B, n, S, \kappa, \Delta, \pi)$
  $b \xleftarrow{\$} \{0, 1\}$
  $ck_T \leftarrow$ (if $b = 0$ then ck_A else ck_B)
  ▷ Generate all other Publishers & Subscribers.
  for $i = 0, \ldots, n$ **do**
    $ck_i \leftarrow \mathcal{N}.\mathsf{GenChannel}(\pi)$
  end for
  ▷ Perform many rounds of the protocols.
  for $t = 0, \ldots$ **do**
    $\Psi_t, \Phi_t \leftarrow \{\}, \{\}$
    ▷ Trigger some Publishers.
    for $i \in \{0, \ldots, n\} \cup \{A, B\}$ **do**
      ▷ Adv. chooses notifications.
      if $\mathcal{A}(i, t, \text{`notify?'}) = 1$ **then**
        $\mu_i \leftarrow \mathcal{N}.\mathsf{Notify}(ck_i, t)$
        $\sigma \leftarrow \mathcal{N}.\mathsf{ProcNotify}(\mu_i, t, \sigma)$
        $\Psi_t \leftarrow \Psi_t \cup \{(i, \mu_i, \sigma)\}$
      end if
    end for
    ▷ Adv. sees all notifications and server state.
    $\mathcal{A}(t, \Psi_t)$
    ▷ Trigger at least u honest Subscribers.
    $\mathcal{Q}_t \leftarrow \{\}$
    $U_t \leftarrow \mathcal{A}(t, u, \text{`GetSubscribers?'})$
    $U_t \leftarrow U_t \cap \{0, \ldots, n\}$
    if $|U_t| < u$ **then**
      return 0
    end if
    ▷ Challenge the target Subscriber.
    if $\mathcal{A}(t, \text{`TargetQuery?'}) = 1$ **then**
      $U_t \leftarrow U_t \cap \{T\}$
    end if
    for all $j \in U_t$ **do**
      $\mathcal{Q}_t \leftarrow \mathcal{Q}_t \cup \mathcal{N}.\mathsf{Query}(ck_j, t, \pi)$
    end for
    for all $\phi_j \in \mathcal{Q}_t$ **do**
      $\rho_j, \sigma \leftarrow \mathcal{N}.\mathsf{ProcQuery}(\phi_j, t, \sigma)$
      $\Phi_t \leftarrow \Phi_t \cup \{(\phi_j, \rho_j, \sigma)\}$
    end for
    ▷ Adv. sees all queries and server state.
    $\mathcal{A}(t, \Phi_t)$
  end for
  return $\mathcal{A}(\text{`Guess?'}) = b$
end procedure

Fig. 2. The Indistinguishable-Notification Experiment.

1 It is inevitable to model a Private Notification System that leaks information. Since the adversary may observe the system for an arbitrary number of past epochs she may learn all other mappings except the challenge one.

subscriber in the protocol run with full knowledge of the secrets it shares with notifiers.

5 AnNotify System

In this section, we first lay out the concrete instantiation of AnNotify as a private notification system. Further, we present the main security theorem of our system and the degree of security obtained for concrete parameters.

5.1 Instantiating AnNotify

The AnNotify system instantiates a Private Notification System using a secure pseudo-random function PRF and an anonymous communication channel, as illustrated in Figure 3.

The $\mathcal{N}$.GenSystem procedure of AnNotify selects a fixed packet length l for the anonymity channel that can comfortably accommodate the transport of a fixed size Bloom filter. Each shard s_i^t, for time $t = 0$, is initialized to an empty Bloom filter. The state of the servers is described by all shards states. The number of users and shards is public through the system parameter π given to all users.

In order to exchange notifications, two (or more) parties must agree on a secret shared key, which we abstract in $\mathcal{N}$.GenChannel, e.g., in an actual system they may use a public key exchange protocol to instantiate it.

When Alice wants to notify Bob about an event, she triggers the $\mathcal{N}$.Notify procedure, which returns the notification identifier, denoted here as μ, computed as a result of a pseudo-random function PRF, keyed with a channel key based on the current epoch. Note that i, the identity of the notification, is returned by $\mathcal{N}$.Notify since the channel is not anonymous.

The server, on receiving a notification, runs $\mathcal{N}$.ProcNotify. In AnNotify, $\mathcal{N}$.ProcNotify first computes to which shard s_i^t the notification should be added and next registers the notification in the shard. If the notification is for a new epoch, a new empty shard is added to the state of the system. This procedure returns a new server state, σ', where shard s_i^t is updated with a new notification. AnNotify does not use the servers' secrets to process the notifications.

In order to query the system, a subscriber uses the $\mathcal{N}$.Query procedure, which recomputes the identifier of the sought event, and uses it to infer which shard ϕ is to be downloaded. The number of shards S is a public information accessible by all users through π. Note, that i – the identifier for the channel – is not returned in ϕ denoting the use of an anonymous channel. Furthermore, one dummy shard is queried. With $\xleftarrow{\$}$ we denote a random selection from a set.

The server receiving a query ϕ simply runs $\mathcal{N}$.ProcQuery and returns the requested shard. The state of the system after $\mathcal{N}$.ProcQuery does not change.

A subscriber after receiving a response checks if the correct Bloom filter is returned and if it contains a notification of interest. If so, the subscriber assumes the notification has occurred.

5.2 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 1 (Section 4.2). 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.

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 proof is presented in Appendix A.1.

The proof depends on a key Lemma 2 (presented in the Appendix), proving a differentially private [13] security bound of IndNotExp where the target subscriber only sends a single query. We use then a generic composition theorem to derive the adversary advantage for r observed queries.

Security Theorem 1 presents a bound on Δ that provides insight about the adversary's advantage based on the security parameters of the system: number of users u, number of shards S and ϵ. However, the presented bound is very loose. Thus, we also present a tighter bound on Δ which we use in the experimental evaluation of the system security properties.

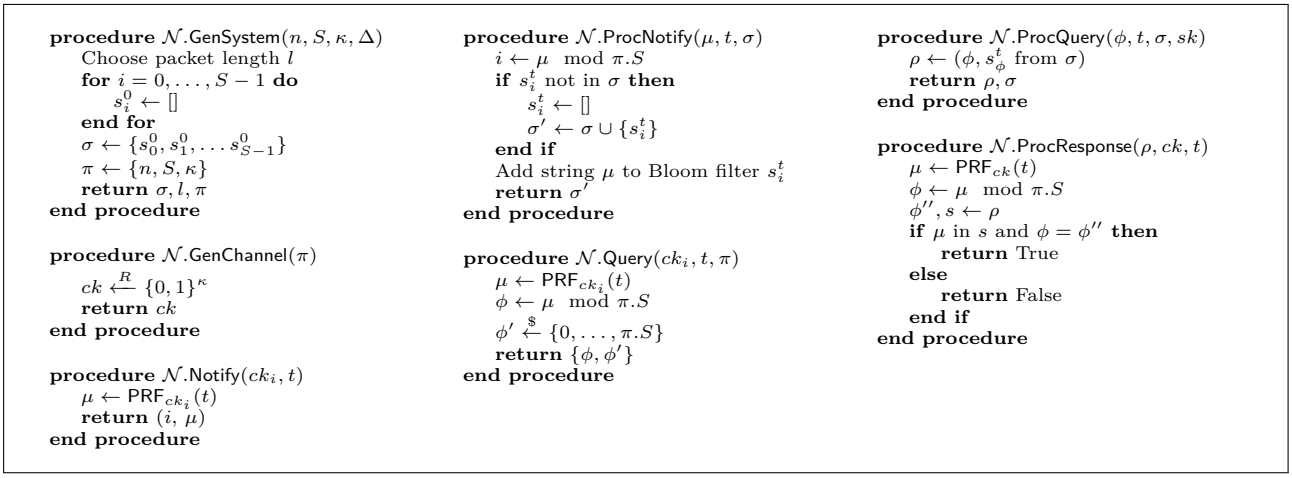


Fig. 3. The concrete instantiation of all algorithms of AnNotify conforming to the interface of a Private Notification System.

Lemma 1. *The AnNotify system is a Δ-private notification system for*

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

where $\epsilon > 0$.

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

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 assumes a threat model in which the adversary does not know the notification shared key. 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.

5.3 Security for concrete parameters

To give a better understanding of the Security Theorem 1 we consider several system parameters, to investigate how scaling of the number of users querying the system influences the number of shards with reference to a required privacy level. The values of δ and Δ define the differential privacy property of the system and the advantage of the adversary in trying to infer the relationship between the publishers and subscribers. We run the simulations using a `Python2.7` script. We first select the threshold values of ϵ, δ such that $\epsilon = 0.2$ and $\delta = \{0.01, 0.001, 0.0001, 10^{-15}\}$. From the dependency between δ and Δ we derive the value of adversary's advantage. For different numbers of users and shards we first compute the the values of Δ following lemma 1 averaged over 100000 samples. Next, we compare the obtained values with the threshold values in order to estimate the maximum number of shards which AnNotify can use without sacrificing the selected security properties. We depict the obtained results in Figure 4.

Each shard stores a Bloom filter responsible for passing the notification between a publisher and subscriber. To make sure, that the *false positive error rate* is small, the parameters of the Bloom filter, length and number of hash functions, should be selected depending on the number of notifying users. The more notifications stored, the larger filter is needed, which influences the

bandwidth consumed. Thus, a larger number of shards helps spread notifications and lower the length of each filter which has to be downloaded by the subscriber.

As the plot depicts, for the fixed parameters of ϵ, δ while the number of honest clients increases, the number of shards which can be used by AnNotify and fulfill at the same time the differential privacy property also increases[2]. This allows to tune the number and size of shards depending on the number of honest users. For example, given 6 million users, we can use 8650 shards, to ensure that the adversary can distinguish two queried notifications only with probability 0.0001, which guarantees that the probability that she correctly deduces the outcome of the challenge game is not higher than $0.5499 + \text{negl}(\kappa)$. For the same values of δ, Δ given 10 million users AnNotify can use 14390 shards.

For parameter $\delta = 10^{-15}$, which results in the adversary's advantage, of correctly guessing after IndNotExp which notification was queried, being $\Delta = 0.0498$, we can observe, that for 9 million querying users, the number of shards which guarantees that the adversary's advantage is bounded by Δ brings 2830. In comparison, for $\delta = 0.01$, the maximum number of shards which can be used to store the clients notifications achieving similar adversary's advantage rises to 32830. This means, that if we assume that for each querying user a notification awaits, there are respectively 3180 and 274 notifications per shard on average. Following this, let us assume that each Bloom filter is using 3 hash functions. The length of a filter storing this number of notifications, which minimizes the type I error, is approximately 100 Bytes for $\delta = 0.01$ and 1.2 KiloBytes for $\delta = 10^{-15}$. Thus, the large number of active users notifications can be easily served by the system achieving both efficiency as well as privacy protection.

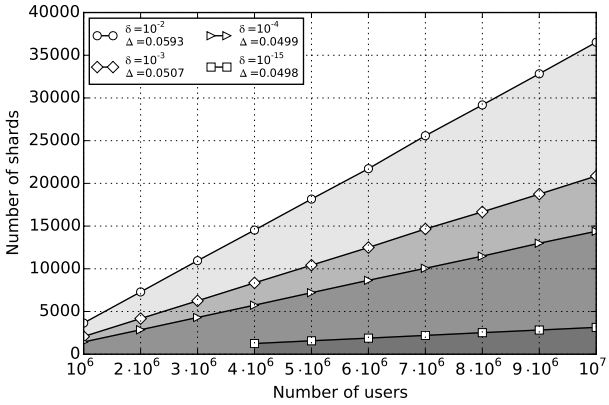


Fig. 4. The maximum number of shards for an increasing number u of requesting users, $\epsilon = 0.2$ and selected values of δ and Δ. Greater number of shards allows better scalability. The values presented on the graph are quantified for $r = 1$.

6 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 servers in AnNotify be S, and the number of servers in the PIR scheme be S'. Since in AnNotify all Bloom filters 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

2 Note that in Theorem 1, because δ depends on ϵ, we obtain an uniform bound for Δ for all values of δ, when ϵ is fixed. The upper bound on δ in Lemma 2 is constant as long as the ratio $\frac{u-1}{S}$ is constant, which explains the linearity of the plot.

Table 1. The cost of action a for particular parameters of the system.

Servers PIR	Shards AnoNotify	Bloom filter size [bits]	Mixing Cost	**Notifications Number**	Error Rate	**a [bits]**	ϵ	δ	Δ
10	$5 \cdot 10^3$	10^4	10	10^7	0.091	$2.9 \cdot 10^5$	0.1	0.0008	0.0258
10	$6.5 \cdot 10^3$	10^4	10	10^7	0.044	$6.1 \cdot 10^5$	0.1	0.0028	0.0278
10	$9 \cdot 10^3$	10^4	10	10^7	0.013	$2.0 \cdot 10^6$	0.1	0.0092	0.0342
10	$16 \cdot 10^3$	10^4	10	10^7	0.0005	$5.8 \cdot 10^7$	0.1	0.0387	0.0637
10	10^4	10^4	10	10^7	0.008	$3.2 \cdot 10^6$	0.1	0.0128	0.0377
10	10^7	10^4	10	10^{10}	0.008	$5.7 \cdot 10^8$	0.1	0.0127	0.0377

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: $a < (S'\sqrt{N \log N} - (l \cdot m_x))2^{\frac{lS}{N}} \log 2$.

In Table 1, we present the threshold cost of action a in AnNotify under different system and security parameters and estimate the values ϵ, δ, Δ using Theorem 1. AnNotify can support thousands or even millions of users at a lower cost than PIR for actions costing kilobytes to megabytes respectively. Furthermore, the security parameters are affected by the ratio of shards to queries which can be kept constant to achieve satisfactory privacy.

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 when a notification is sent and when it can be read is $|e|+t$, where t is the 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.

In AnNotify, although all notifications received in an epoch are available at the start of the next epoch, if a publisher-subscriber epoch is much smaller than the server epoch, the subscriber has to request many times to check if a notification was received in any of the possible publisher-subscriber epochs. Clearly the smaller the publisher-subscriber epoch length, the more often a subscriber has to request the system for notifications.

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.

7 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, and in particular, discuss design decisions to improve performance, and compare it with other systems.

7.1 Design for Distributing Load

In epoch-based presence mechanisms, such as DP5 [5], we expect a high load of queries at the beginning of every epoch. At that time, clients both update their presence and retrieve the presence of their friends. To distribute the load, AnNotify follows a different approach without increasing average latency.

An AnNotify client periodically sends a single request to both report a notification and perform a query. Assume epochs of $|e|$ minutes each. Every AnNotify client uniformly chooses an offset time from $(0, |e|)$ from the beginning of the epoch and at that time sends the combined notification and query to the shard.

To implement the AnNotify scheme, each AnNotify server maintains two Bloom filters for each shard it controls, the *current* and the *next*. Upon receiving a notification the server always adds elements to the *next* Bloom filter and returns the *current* filter. At the end of every epoch, the *next* becomes the *current* and *next* is set to be an empty Bloom filter.

The data in the *current* Bloom filter that users receive is updated to $|e|$ minutes before the response is generated on average. This latency is the same, on av-

erage, as for clients accessing the server at the beginning of the epoch (like in DP5): since the client sends the requests uniformly within each epoch, there is an average delay of $\frac{|e|}{2}$ minutes from the beginning of the current epoch. The subscriber is interested in notifications from a previous epoch, which were also posted at an uniform time within the previous epoch. Therefore, there is additional delay of $\frac{|e|}{2}$ minutes on average. Hence, the total delay is a single epoch ($|e|$ minutes) on average, as predicted by the theoretical latency analysis.

7.2 Implementation & Infrastructure

We implement AnNotify servers as a web-server that subscribers may easily access through the most popular anonymity network today, Tor [12]. 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. Leveraging this, 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.

7.3 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.

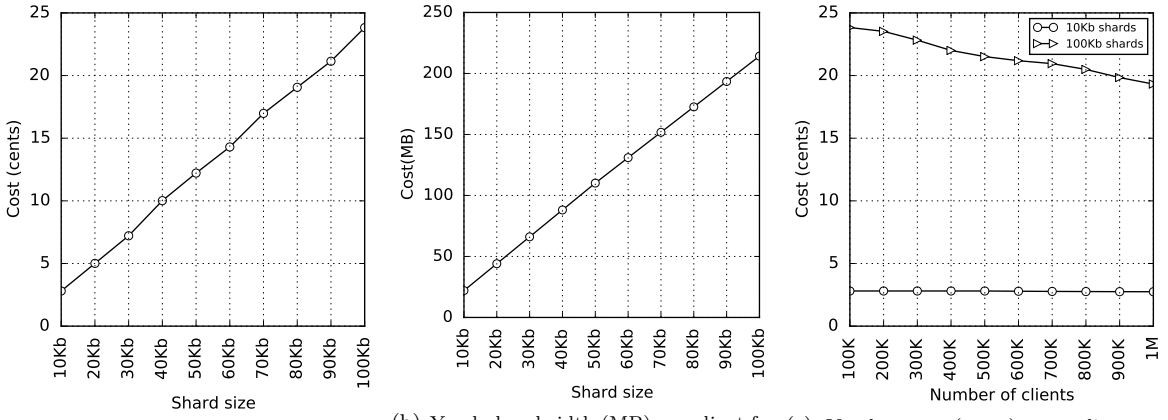
We estimated the cost of running AnNotify in the Amazon cloud. The main factor in the cost calculation was the bandwidth that increases linearly as a function of the shard size. However, the bandwidth cost per byte decreases as the system consumes more bandwidth, e.g., for larger shards and for more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

Figure 5 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 5(a)

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

Fig. 5. 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.

shows that the yearly cost per client ranges from a few cents (shards of 10Kb) to less than a quarter (shards of 100Kb). Figure 5(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 5(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 $1M$ 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 $1M$ users.

7.4 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 5(c)) with significantly lower yearly cost than DP5, of a few cents.

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 leakage Δ, that we compute empirically (to get a tight estimate, see appendix A.4), added to the probability f^ℓ that all mix-servers are corrupt.

Bandwidth. Figure 6 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.

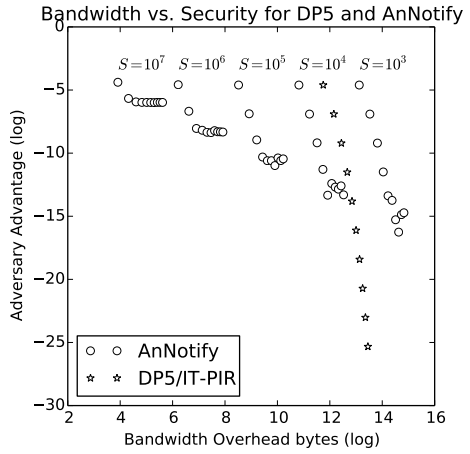


Fig. 6. Security versus Bandwidth comparison for AnNotify and DP5/IT-PIR, for different parameters $\ell \in \{2, \ldots, 11\}$ and $S \in \{10^3, \ldots, 10^8\}$. Database of $u = 10^9$ bits and users, and fraction of corrupt nodes $f = 10\%$. We observe that AnNotify is orders of magnitude cheaper when some leakage may be tolerated (adversary advantage $e^{-5} \ldots e^{-10}$). (Smaller is better on both axis.)

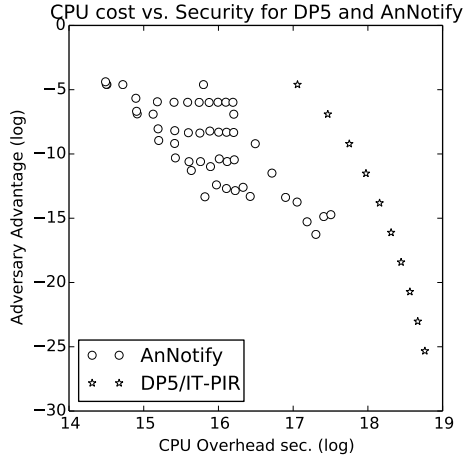


Fig. 7. Security versus CPU cost comparison for AnNotify and DP5/IT-PIR, for different parameters $\ell \in \{2, \ldots, 11\}$ and $S \in \{10^3, \ldots, 10^8\}$. Database of $u = 10^9$ bits and users, and fraction of corrupt nodes $f = 10\%$. We observe that AnNotify requires less processing time than IT-PIR for comparable security levels. (Smaller is better on both axis.)

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, 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 7 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.

8 AnNotify Extensions

In this section, we present possible extensions of AnNotify. First, we discuss a continuous variant of AnNotify which does not require operating in epochs. Next, we present how AnNotify design can be modified in order to serve as a presence system. Last but not least, we argue that AnNotify guarantees privacy properties while supporting broadcast notifications.

The Continuous AnNotify Design. The design of AnNotify system outlined in Section 3 results in a long waiting period of one epoch between when a notification is published and when a notification may be queried, depending on the epoch length at the server. We present a variant of the AnNotify system that does not require the server and publisher-subscriber epochs to be synchronized and allows the servers to serve notifications corresponding to an arbitrary moving time-window: over time, each server constructs a Bloom filter that contains all notifications within a fixed past period. It does so incrementally, and may also wish to modify the Bloom filter parameters to ensure a fixed error rate – which adds security considerations.

In the simple AnNotify design each shard is represented by a distinct standard Bloom filter structure for each server side fixed term epoch. In the continuous design notifications are instead added and deleted continuously to a single Bloom filter as events are included and excluded from a time-window. The length of this time window is application dependent, and set at the server that manages the shards.

Standard Bloom filters cannot be efficiently updated in the desired manner to reflect additions and deletions, so we instead use a counting Bloom filter[4, 15]: the server keeps a count of the number of items contributing to each bin of the Bloom filter. Adding

6 Using the Python `sphinxmix` package.

an element to the Bloom filter involves increasing the appropriate bins by one, and removing an element decreasing them again by one.

The counting Bloom filter can then be quantized per bin to take no more space than a traditional Bloom filter when served to clients. Bins with a zero count are assigned a 'zero' element, and non-zero count bins a 'one'. This resulting Bloom filter is equally susceptible to false positives. We note that the cost of adding or removing elements to the Bloom filter is fixed, and therefore managing the data structure over time is a constant time per notification received.

AnNotify as a presence system. In this section, we describe how AnNotify can be used as a presence system, to transmit a small amount of information from the publisher to the subscriber.

In this variant, we do not use Bloom filters, but instead each server stores the received notifications as a list within each shard. 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 $\mathsf{AES\text{-}GCM}$) 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 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 would 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 mix network or other anonymity infrastructure, and untrusted servers holding shards. Given our evaluation results, relating the adversary advantage to performance, such designs may be competitive 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, and 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 major 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.

9 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 adding anonymity to 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 al-

low 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 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 [21]. 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 [18], 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 [17, 28].

In all applications, AnNotify allows preserving the privacy of users, by hiding the relationships between users and the notifications they receive or send. 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 [20, 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]. However, to fully benefit from AnNotify it would have to be extended to a group notification setting, ensuring only one publisher may post a notification, an multiple subscribers can receive it privately. This is a challenging area for future work.

10 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] at the expense of latency. 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. It allows the pseudonym holders to receive messages, while ensuring unlinkability among messages and their recipients with forward secrecy.

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, 19]. IT-PIR is a multiple server PIR vari-

ant, 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. Both traditional IT-PIR as well as its new modifications, require operators of a system to decide, whether they favor strong privacy with large performance costs, through using multiple servers; or low performance costs with 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. As discussed previously, applying IT-PIR significantly increases the bandwidth and subsequently the communication and performance cost, this presents difficulties when scaling to a large user-base. 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. In comparison to this scheme, AnoNotify also guarantees that the responses from the server are always the same length and the size of the response does not reveal the number of stored elements.

11 Conclusions

AnNotify provides efficient and private notifications in a scalable manner, compared with previous approaches like DP5 [5] that struggled to scale past 1 million users, with credible cost estimates when it comes to scaling to even billion of 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, that today benefit from no protections at all.

Besides securing notifications, the AnNotify design, provides a couple of important insights into general privacy engineering. First, 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.

Second, we show that AnNotify can be extended to provide generic PIR services, with minor alterations to our main Security Theorem. Since even IT-PIR, from a security engineering perspective, may fail (if all servers are corrupt), PIR schemes inspired from the AnNotify design and anonymous channels, may be more competitive in terms of performance that those proposed so far. Pursuing this research direction would allow wider deployment of private querying in general.

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. We would also like to thank the anonymous PoPETS 2017.02 reviewers for their suggestions, which helped to improve this paper.

References

[1] BBC. "Yo app warns Israeli citizens of missile strikes". Online, July 2014.

[2] G. Bianchi, L. Bracciale, and P. Loreti. Better than nothing. privacy with Bloom filters: To what extent? In *Privacy in Statistical Databases - PSD 2012, Palermo, Italy, September 26-28, 2012.*, pages 348–363, 2012.

[3] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[4] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting Bloom filters. In *Algorithms – ESA*, UK, 2006.

[5] N. Borisov, G. Danezis, and I. Goldberg. DP5: A private presence service. *PoPETs*, 2015(2):4–24, 2015.

[6] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[8] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[9] S. Cohen and Y. Matias. Spectral Bloom filters. In *Conference on Management of Data (SIGMOD)*, SIGMOD '03, pages 241–252, New York, NY, USA, 2003. ACM.

[10] G. Danezis and I. Goldberg. 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*, pages 269–282, 2009.

[11] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *USENIX Security 12*, pages 269–283, 2012.

[12] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *USENIX Security*, pages 303–320. USENIX, 2004.

[13] C. Dwork. Differential privacy. *ICALP*, 2006.

[14] U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, pages 1054–1067, USA, 2014. ACM.

[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, June 2000.

[16] H. Federrath, K. Fuchs, D. Herrmann, and C. Piosecny. Privacy-preserving DNS: Analysis of broadcast, range queries and mix-based protection methods. In *Computer Security - ESORICS 2011, Leuven, Belgium, September 12-14, 2011.*, 2011.

[17] S. Geravand and M. Ahmadi. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks*, 57(18):4047–4064, 2013.

[18] T. Gerbet, A. Kumar, and C. Lauradoux. A privacy analysis of google and yandex safe browsing. Technical Report Research Report RR-8686, INRIA, Sept. 2015.

[19] I. Goldberg. Improving the robustness of private information retrieval. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 131–148. IEEE, 2007.

[20] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[21] Google. Google transparency report - making the web safer, June 2014.

[22] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[23] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[24] B. Laurie. Apres-a system for anonymous presence, 2004.

[25] Y. Lu and G. Tsudik. Towards plugging privacy leaks in the domain name system. In *Peer-to-Peer Computing*, pages 1–10. IEEE, 2010.

[26] D. Mazières and M. F. Kaashoek. The design, implementation and operation of an email pseudonym server. In *Conference on Computer and Communications Security*, CCS '98, pages 27–36, 1998.

[27] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[28] S. D. Paola and D. Lombardo. Protecting against DNS reflection attacks with Bloom filters. In T. Holz and H. Bos, editors, *DIMVA*, 2011.

[29] T. T. project. Tor Metrics. https://metrics.torproject.org/, April 2016.

[30] C. E. Rothenberg, C. Macapuna, F. L. Verdi, and M. F. Magalhães. The deletable Bloom filter: A new member of the Bloom family. *CoRR*, abs/1005.0352, 2010.

[31] L. Sassaman, B. Cohen, and N. Mathewson. 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, pages 1–9, 2005.

[32] H. Shulman. Pretty bad privacy: Pitfalls of DNS encryption. In *Workshop on Privacy in the Electronic Society, WPES 2014*, 2015.

[33] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *Computer & communications security (ACM CCS)*, pages 299–310. ACM, 2013.

[34] R. R. Toledo, G. Danezis, and I. Goldberg. Lower-cost ϵ-private information retrieval. *Proceedings on Privacy Enhancing Technologies*, 2016(4):184–201, 2016.

[35] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[36] Wikipedia. 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, defined in Definition 1, with an advantage Δ 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 1, including the IndNotExp experiment (Game_0). We first note that in the concrete protocols $\mathcal{N}.\mathsf{Notify}$ and $\mathcal{N}.\mathsf{Query}$ act on notification IDs generated using a pseudo-random function (PRF) keyed with an unknown key to the adversary and the epoch number ($ID^t = \mathsf{PRF}_{ck}(t)$). Thus, from the adversaries point of view, the IDs 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 2 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 then concludes the proof by showing this differentially private property can be translated to a concrete adversary advantage Δ gained by observing many epochs. □

A.2 Lemma 2 from Section 5.2

Let us first define the following notation

Definition 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* $\Pr[(X_A, X_B) \in \mathcal{A}] \geq 1-\delta$.

Using this definition, let us prove the following lemma.

Lemma 2. *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*

$$\delta \leq \exp\left(-\frac{(u-1)}{4S}\right) \tag{1}$$

$$+ \exp\left(-\frac{(u-1)}{2S} \tanh^2\left(\frac{\epsilon}{2}\right)\right) \tag{2}$$

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

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.

Proof. We define as S_A, S_B the events that either shard s_A or s_B was queried. For a mapping function F, which maps the notifications identifiers to the storing shards, such that $F(A) = s_A$, we have

$$\begin{aligned}&\Pr[X_A = x_A, X_B = x_B | I_A] \\ &= \Pr[X_A = x_A, X_B = x_B | I_A, S_A].\end{aligned} \tag{3}$$

Using the properties of the conditional probability[8] we obtain the following equality (proof in Lemma 3 in Appendix A.3), and using (3) we have:

$$\begin{aligned}&\Pr[X_A = x_A, X_B = x_B | I_A, S_A] \\ &= \Pr[X_A = x_A, X_B = x_B | S_A] \\ &\qquad\downarrow \\ &\Pr[X_A = x_A, X_B = x_B | I_A] \\ &= \Pr[X_A = x_A, X_B = x_B | S_A].\end{aligned} \tag{4}$$

Now, we can prove that the events when either notification A or B is requested in the challenge is (ϵ, δ) - *differentially private*, so the adversary who wants to infer which notification is queried is not able to distinguish this two events with significant probability.

We define $k = x_A + x_B - 1$. The probabilities that either notification A or B is request are denoted as

$$\begin{aligned}\Pr[X_A = x_A, X_B = x_B | S_A] =& \binom{u-1}{k} \left(\frac{2}{S}\right)^k \left(\frac{S-2}{S}\right)^{u-k-1} \\ &\cdot \binom{k}{x_A - 1} \left(\frac{1}{2}\right)^{x_A - 1} \left(\frac{1}{2}\right)^{k - x_A + 1} \\ \Pr[X_A = x_A, X_B = x_B | S_B] =& \binom{u-1}{k} \left(\frac{2}{S}\right)^k \left(\frac{S-2}{S}\right)^{u-k-1} \\ &\cdot \binom{k}{x_B - 1} \left(\frac{1}{2}\right)^{x_B - 1} \left(\frac{1}{2}\right)^{k - x_B + 1}.\end{aligned} \tag{5}$$

The above equality results from the binomial distribution[9]. Thus, we have

$$\Pr[X_A = x_A, X_B = x_B | S_A] = \frac{x_A}{x_B} \Pr[X_A = x_A, X_B = x_B | S_B]. \tag{6}$$

So we would like to ensure, that $\frac{x_A}{x_B} \leq e^{\epsilon}$, which implies $x_B \geq e^{-\epsilon} x_A$.

We remind that the expected value of the binomial distribution $Bin(n, p)$ is $E[X] = np$. Thus, the expected value of queries sent to shard S_A and S_B is $E[X] = \frac{2(u-1)}{S}$. If the number of queries destined to S_A and S_B is smaller than the expected value, the adversary observes very *sparse* hiding set for the target query and has bigger chances to guess correctly if either notification from A or B was queried. Thus, we define $C = \frac{2(u-1)}{S} - \gamma$, where $\gamma = \tau \cdot \frac{2(u-1)}{S}$ and $\tau \in (0, 1)$. We define the value of δ (related to the events when it is easy to distinguish the two observations in our challenge) as below

$$\begin{aligned}\delta =& \underbrace{\Pr[\overset{(I)}{X_A + X_B \leq C}]}_{\mathrm{Bin}\left(u-1, \frac{2}{S}\right)} \\ &+ \underbrace{\Pr[X_B \leq e^{-\epsilon} X_A | \overset{(II)}{X_A + X_B \geq C}]}_{\mathrm{Bin}\left(C, \frac{1}{2}\right)} \underbrace{\Pr[X_A + X_B \geq C]}_{\mathrm{Bin}\left(u-1, \frac{2}{S}\right)} \\ \delta =& \Pr[\overset{(I)}{X_A + X_B \leq C}] + \Pr[X_B \leq e^{-\epsilon} X_A \wedge \overset{(II)}{X_A + X_B \geq C}]\end{aligned} \tag{7}$$

We recall the Chernoff bound [27] for a random variable X, which is a sum of independent variables with Bernoulli distribution, defined as

$$\Pr[X \leq (1-d)E[X]] \leq e^{\frac{-E[X]d^2}{2}}.$$

First, we estimate the part (I) of equation (7)

$$\begin{aligned}\Pr\left[X_A + X_B \leq C\right] &\overset{X = X_A + X_B}{=} \Pr\left[X \leq C\right] \\ &= \Pr\left[X \leq \frac{2(u-1)}{S} - \gamma\right] \\ &= \Pr\left[X \leq (1-\tau)\frac{2(u-1)}{S}\right] \\ &\leq \exp\left(-\frac{2(u-1)\tau^2}{2S}\right) \\ &= \exp\left(-\frac{(u-1)\tau^2}{S}\right).\end{aligned} \tag{8}$$

Now, we compute part (II) of equation 7 by first deriving an upper bound using the Hoeffding's inequality [22].

$$\begin{aligned}&\Pr[X_B \leq e^{-\epsilon} X_A \wedge X_A + X_B \geq C] \\ &= \sum_{i=C}^{u-1} \Pr[X_B \leq e^{-\epsilon} X_A \wedge X_A + X_B = i] \\ &= \sum_{i=C}^{u-1} \Pr[X_B \leq e^{-\epsilon} X_A | X_A + X_B = i] \Pr[X_A + X_B = i] \\ &\leq \sum_{i=C}^{u-1} \Pr[X_B \leq e^{-\epsilon} X_A | X_A + X_B = C] \Pr[X_A + X_B = i] \\ &= \Pr[X_B \leq e^{-\epsilon} X_A | X_A + X_B = C] \Pr[X_A + X_B \geq C]\end{aligned} \tag{9}$$

8 $\Pr[A|C,B] \cdot \Pr[C|B] = \Pr[A,C|B] \implies \sum_C (\Pr[A|C,B] \cdot \Pr[C|B]) = \Pr[A|B]$, for the events A, B, C, such that $\Pr[C,B] > 0, \Pr[B] > 0$.

9 The probability distribution function of the binomial distribution $\mathrm{Bin}\left(n, \frac{1}{p}\right)$ is $\Pr[X = k] = \binom{n}{k} \left(\frac{1}{p}\right)^k \left(1 - \frac{1}{p}\right)^{n-k}$.

Thus, $X_B \leq e^{-\epsilon}(C - X_B)$, which implies $X_B \leq \frac{e^{-\epsilon}C}{1+e^{-\epsilon}}$. Applying the Hoeffding's inequality we obtain the following

$$\begin{aligned}
&\Pr[X_B \leq e^{-\epsilon}X_A \wedge X_A + X_B \geq C] \\
&= \Pr[X_B \leq \frac{e^{-\epsilon}C}{1+e^{-\epsilon}}] \Pr[X_A + X_B \geq C] \\
&\leq \exp\left(-\tfrac{(u-1)(1-\tau)}{S}\left(1 - 2\tfrac{e^{-\epsilon}}{1+e^{-\epsilon}}\right)^2\right) \\
&\quad \cdot (1 - \underbrace{\sum_{k=0}^{C-1} \tbinom{u-1}{k} \left(\tfrac{2}{S}\right)^k \left(1 - \tfrac{2}{S}\right)^{u-1-k}}_{\texttt{CDF}[u-1, \frac{2}{S}, C-1]}) \\
&\leq \exp\left(-\tfrac{(u-1)(1-\tau)}{S}\left(1 - 2\tfrac{e^{-\epsilon}}{1+e^{-\epsilon}}\right)^2\right).
\end{aligned} \tag{10}$$

As $\texttt{CDF}[n, p, x]$ we denote a cumulative distribution function of binomial distribution.
Taking these two equations together, we obtain the following bound for the value of δ for $\tau = \frac{1}{2}$

$$\delta \leq \exp\left(-\frac{(u-1)}{4S}\right) + \exp\left(-\frac{(u-1)}{2S}\tanh^2\left(\frac{\epsilon}{2}\right)\right) \tag{11}$$

The above bound gives us the estimation of the value of δ, which bounds the probability of very rare events which can destroy our differential privacy guarantee. Note, that we have a dependency between δ and ϵ in this equation, so we can select both values to work the best for us.

The presented estimation is however a crude upper bound, thus we next present the precise computation of the probability value, based on the cumulative distribution function, which we use in the experimental evaluation of the security properties in Section 5.3.

$$\begin{aligned}
&\Pr[X_B \leq e^{-\epsilon}X_A \wedge X_A + X_B \geq C] \\
&= \sum_{i=C}^{u-1} \Pr[X_B \leq \frac{ie^{-\epsilon}}{1+e^{-\epsilon}}] \Pr[X_A + X_B = i] \\
&= \sum_{i=C}^{u-1} \left(\sum_{k=0}^{\alpha} \binom{i}{k}\left(\frac{1}{2}\right)^i\right) \binom{u-1}{i}\left(\frac{2}{S}\right)^i \left(1 - \frac{2}{S}\right)^{u-1-i} \\
&= \sum_{i=C}^{u-1} \texttt{CDF}\left[i, \frac{1}{2}, \alpha\right] \binom{u-1}{i}\left(\frac{2}{S}\right)^i \left(1 - \frac{2}{S}\right)^{u-1-i}
\end{aligned} \tag{12}$$

where $\alpha = \frac{ie^{-\epsilon}}{1+e^{-\epsilon}}$. Hence, we obtain the following value of δ.

$$\begin{aligned}
\delta =& \texttt{CDF}\left[u-1, \frac{2}{S}, C\right] \\
&+ \sum_{i=C}^{u-1} \texttt{CDF}\left[i, \frac{1}{2}, \alpha\right] \binom{u-1}{i}\left(\frac{2}{S}\right)^i \left(1 - \frac{2}{S}\right)^{u-1-i}.
\end{aligned} \tag{13}$$

We can now show, that we can derive the definition of (ϵ, δ) - *differential privacy* from the above result. Let us denote $Z = [X_A, X_B | I_A]$. Thus, from the law of total probability we have that

$$\Pr[Z] = \Pr[Z|G]\Pr[G] + \Pr[Z|N]\Pr[N]$$

where as G we denote events, when values drawn for X_A, X_B satisfy the inequality $\Pr[X_A, X_B|I_A] \leq e^{\epsilon}\Pr[X_A, X_B|I_B]$ and as N we denote otherwise. Thus, since we know, that $\Pr[N] \leq \delta$ and both $\Pr[G]$ and $Pr[Z|N]$ can be upper bounded by 1, using above result we obtain

$$\begin{aligned}
\Pr[Z] &\leq e^{\epsilon}\Pr[X_A, X_B|I_B] + \delta, \\
\Pr[X_A, X_B|I_A] &\leq e^{\epsilon}\Pr[X_A, X_B|I_B] + \delta.
\end{aligned}$$

□

Lemma 3. *For random variables X_A, X_B and events I_A, I_B, S_A, S_B defined as in Lemma 2 we have the following dependency*

$$\Pr[X_A, X_B|I_A, S_A] = \Pr[X_A, X_B|S_A].$$

Proof. From conditional probability properties and the fact that $\Pr[S_A] = \frac{1}{2}$, $\Pr[I_A, S_A] > 0$ we can write that

$$\sum_i (\Pr[X_A, X_B|i, S_A] \cdot \Pr[i|S_A]) = \Pr[X_A, X_B|S_A].$$

The sum of the probabilities over requested notifications can be considered as a sum of the probabilities for the notification i which map to shard s_A and those who do not. As N_A we denote a set of notifications whose identifiers map to shard s_A. Following this, we can present the previous equation as

$$\begin{aligned}
&\sum_{i \in N_A} \Pr[X_A, X_B|i, S_A]\Pr[i|S_A] \\
&+ \sum_{i \notin N_A} \Pr[X_A, X_B|i, S_A]\Pr[i|S_A] \\
&\quad = \Pr[X_A, X_B|S_A].
\end{aligned}$$

Because $\Pr[i|S_A] = 0$ for each $i \notin N_A$ we have

$$\Pr[X_A, X_B|I_A, S_A] \cdot \sum_{i \in N_A} \frac{1}{|N_A|} = \Pr[X_A, X_B|S_A]$$
$$\Rightarrow \Pr[X_A, X_B|I_A, S_A] = \Pr[X_A, X_B|S_A].$$

□

A.3 Lemma 4 from Section 5.2

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. In each round the shards for notifications are selected at random, thus each round the target notification is stored in a different shard. As $O_i = (X_A^i, X_B^i)$ we denote the observation of number of queries observed coming to shard s_A and s_B respectively in round i.

Lemma 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:

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

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

Proof. From Lemma 2 we know, that a differentially private bound for a single round holds for the probability an adversary observes volumes of the shards resulting from events $S_{b=0}, S_{b=1}$ (with some probability $1 - \delta$). From the fact, that the observations in each round are independent we obtain

$$\Pr[O_1, O_2, \ldots, O_r|S_{b=0}] = \prod_{i=1}^{r} \Pr[O_i|S_{b=0}],$$
$$\prod_{i=1}^{r} \Pr[O_i|S_{b=0}] \leq e^{r\epsilon} \prod_{i=1}^{r} \Pr[O_i|S_{b=1}],$$

On the basis that since $\Pr[S_{b=0}] = \Pr[S_{b=1}] = \frac{1}{2}$

$$\prod_{i=1}^{r} \Pr[O_i|S_{b=0}] \Pr[S_{b=1}] \leq e^{r\epsilon} \prod_{i=1}^{r} \Pr[O_i|S_{b=1}] \Pr[S_{b=0}],$$
$$\prod_{i=1}^{r} \Pr[S_{b=0}|O_i] \leq e^{r\epsilon} \prod_{i=1}^{r} \Pr[S_{b=1}|O_i],$$
$$\Pr[S_{b=0}|O_1, \ldots, O_r] \leq e^{r\epsilon} \Pr[S_{b=1}|O_1, \ldots, O_r]$$

Following this, we obtain

$$\frac{1}{2} + \Delta_\epsilon \leq e^{r\epsilon}\left(\frac{1}{2} - \Delta_\epsilon\right)$$
$$\Delta_\epsilon \leq \frac{e^{r\epsilon} - 1}{2\left(e^{r\epsilon} + 1\right)} = \frac{1}{2}\tanh\left(\frac{r\epsilon}{2}\right).$$

Given value of Δ_ϵ, we can compute the adversary's overall advantage after r rounds. The total value of Δ, which is the adversary's overall advantage of guessing successfully is bounded as

$$\Delta \leq \left(1 - \sum_{i=1}^{r} \delta\right) \cdot \frac{1}{2}\tanh\left(\frac{r\epsilon}{2}\right) + \sum_{i=1}^{r} \delta. \tag{14}$$

since the differential privacy holds in each round with probability $1 - r\delta$ or otherwise, if in at least one round ($r\delta$) the distribution of query volumes does not guarantee the differential privacy, we can assume that the adversary automatically can guess correctly. Since $(1 - r\delta) \leq 1$ we loose the bound of Δ by

$$\Delta \leq \frac{1}{2}\tanh\left(\frac{r\epsilon}{2}\right) + r\delta$$

Value of Δ is the advantage of the adversary in successfully guessing the value of bit b over a random guess. □

A.4 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. For the comparison with DP5 in Section 7.4 we instead estimate $\hat{\Delta}$ the advantage of the adversary 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. 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 rare and has never occurred.

Finally, we estimate $\hat{\epsilon}$ as the mean adversary advantage for all rounds with positive $\vec{x_B}$:

$$\hat{\epsilon} = 1/I \cdot \sum_i \log x_A^i / x_B^i,$$

where I denotes the number of samples. This is the log of the Geometric mean of the advantages for each epoch. 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. Thus, we use it to accurately compare security and performance with DP5.