Computer Communications 211 (2023) 178–192

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Privacy-friendly statistical counting for pedestrian dynamics

Valeriu-Daniel Stanciu [a,*], Maarten van Steen [a], Ciprian Dobre [b], Andreas Peter [c,a]

[a] University of Twente, Drienerlolaan 5, Enschede, 7522 NB, The Netherlands
[b] University Politehnica of Bucharest, Splaiul Independentei 313, Bucharest, 060042, Romania
[c] University of Oldenburg, Ammerländer Heerstr. 114-118, Oldenburg, 26129, Germany

ARTICLE INFO

Keywords:
Crowd monitoring
Pedestrian dynamics
Statistical counting
Privacy-preservation
Bloom filters
Homomorphic encryption

ABSTRACT

Relying on Wi-Fi signals broadcasted by smartphones became the de-facto standard in the domain of pedestrian crowd monitoring. This method got the edge over other traditional means owing to the fact that insights are built upon data which uniquely identifies individuals and, thus, allows highly accurate crowd profiling over time. On the other hand, handling such uniquely identifying data in such a way that it does not expose the sensed individuals to potential privacy infringements proves to be a difficult task. Although several protection techniques were proposed, they yield data which, combined with other external knowledge, can still be used for tracing back to specific individuals. To address this issue, we propose a construction which protects the short-term storage and processing of privacy-sensitive Wi-Fi detections under strong cryptographic guarantees and makes available in the clear, as end results, only statistical counts of crowds. To produce these statistical counts, we make use of homomorphically encrypted Bloom filters as facilitators for oblivious set membership testing under encryption. We implement the system and perform evaluation on both simulated data and a real-world crowd-monitoring dataset, demonstrating that it is feasible to achieve highly accurate statistical counts in a privacy-friendly way.

1. Introduction

Performing Wi-Fi-based crowd monitoring for understanding pedestrian dynamics has become commonplace practice, as being able to observe the crowd behavior is cornerstone for successfully managing crowded public areas. Wi-Fi scanners installed in public spaces gather signals broadcasted by devices carried by individuals. By leveraging such signals, interested parties can estimate the size of crowds near those scanners, as well as the size of the flows developing between them. Such information proved to be useful on numerous occasions, for example for analyzing mass events [1,2], identifying travel patterns [3], uncovering social interactions [4,5] or even preventing critical situations [6,7].

Dealing with data related to crowds has always been a sensitive matter, regardless the technique used for monitoring, mainly because insights are built upon the people making up these crowds, people who have privacy concerns. In Wi-Fi-based crowd monitoring, signals gathered by scanners contain unique identifiers, i.e. MAC addresses, corresponding to devices carried by individuals. Detecting these identifiers at different locations over time allows the system to build up crowd-level knowledge on pedestrian dynamics based on the movement patterns of individuals. In such a system, an individual could be uniquely re-identified from data bearing his identifier and have his every move followed, infringing thus his privacy.

In an effort to prevent such situations from happening, sets of rules were proposed to clearly regulate the process, like, for example, the General Data Protection Regulation [8] (GDPR) in the EU. However, existing data protection strategies for Wi-Fi based crowd monitoring proved on several occasions not to provide a proper protection for the individuals being sensed. Currently used strategies are based on replacing real identifiers with pseudonyms obtained either by hashing the MAC addresses with a one-way hash function, encrypting them with a deterministic encryption scheme or assigning them a random token generated by a cryptographically secure pseudorandom number generator. Pseudonyms still allow tracking over time and space, as well as creating individual profiles which, under certain conditions, for example when external knowledge is available, remain susceptible to re-identification. As a result, organizations doing crowd monitoring ended up facing difficult challenges while delivering their services. Many of them halted their activities [9–11], while others are being fined due to privacy-related incidents [12].

The aim of a crowd-monitoring system for pedestrian dynamics is to provide insights on crowds in the form of statistical counts. In the process of building such aggregated information, privacy-sensitive data

* Corresponding author.
E-mail address: v.stanciu@utwente.nl (V. Stanciu).

https://doi.org/10.1016/j.comcom.2023.09.009
Received 30 March 2023; Received in revised form 24 July 2023; Accepted 4 September 2023
Available online 6 September 2023

of individuals has to be used. Following data protection as a goal and data minimization as a way to achieve it, we envision a system that offers statistical counts as the only accessible information in the clear while protecting the privacy-sensitive data of individuals at rest and during processing.

Searching for a method to facilitate oblivious crowd observance over time, in [13] we investigated the use of Bloom filters (BFs), i.e. probabilistic data structures supporting set operations, together with homomorphic encryption (HE), i.e. a type of encryption that allows performing operations on encrypted data. Preliminary experiments indicated that such a scheme can indeed be suitable for our goal, allowing counting over encrypted representations of sets or intersections of sets of devices without revealing what is being counted.

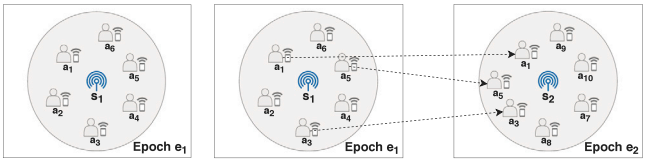
Having shown that by combining BFs with HE it is possible to provide statistical counts while protecting the data of individuals, in this article we address the problem of actually designing a crowd-monitoring system based on the proposed principles, as well as what has to be done in order to make the deployment of such a system feasible. The contributions of this article are summarized as follows.

- We propose a crowd-monitoring system that can produce statistical counts as a service for interested consumers while protecting the privacy-sensitive data of sensed individuals. The system is secure against honest-but-curious adversaries and it allows counting crowds at one location, as well as counting the crowd flow between locations.
- We carry out an implementation of the system using Raspberry Pi as a typical sensing device and two different server configurations (i.e. a laptop and a more powerful cloud server) as operators under encryption, to assess the feasibility of our solution. We deploy the system, trialing different setup parameters, and analyze its performance when faced with a whole range of crowds.
- We perform a thorough evaluation using simulated data, to explore the potential of our solution to estimate statistical counts, as well as using real-world data from an open-air festival consisting of 26 million detections, to see how the system performs when dealing with real detections generated by actual pedestrian movements. Statistical counts regarding footfall are estimated with an accuracy of at least 97.2% when analyzing the most crowded area of the festival. 88.5% of the statistical counts on crowd flows happening during the festival on a circulated street have an accuracy of at least 90%, while 98.7% of the estimations are less than 3 devices away from the real counts.

The rest of this article is organized as follows. Section 2 introduces the system model, including crowd-monitoring formalities, involved actors and security requirements. Section 3 presents our construction instantiating the system model by combining BFs with HE. In Section 4 we perform an evaluation of the system on simulated data to get an understanding of how well statistical counts can be estimated. Section 5 presents an implementation of the system, together with a performance analysis. Section 6 evaluates our system on real-world data from a mass event, followed by a discussion in Section 7. Section 8 reviews the related work and, finally, Section 9 concludes the article.

2. System model

Crowd-monitoring systems are usually deployed to provide an understanding of the pedestrian dynamics happening in crowded public spaces. Hence, the world of such a system is represented by a public space where crowds of people are expected. In this space, a technical infrastructure is usually installed to collect data about the crowd. Based on that sensed data, useful information is built in the form of crowd-monitoring insights. Considering that for building these insights data related to people is handled by multiple entities, the whole process should be governed by some clearly stated rules to ensure that the privacy of the sensed individuals is protected. We start by first introducing the Wi-Fi-based crowd-monitoring environment. Then, we model the insights offered by the system as statistical counts for pedestrian dynamics. Eventually, we describe the actors involved in the process, together with requirements such that the privacy-sensitive data of individuals is protected.



(a) (b)

Fig. 1. Situations encountered in pedestrian dynamics: (a) *footfall* and (b) *crowd flow*.

2.1. Crowd-monitoring environment

Typically, setting up a Wi-Fi-based crowd-monitoring system starts by installing a set of Wi-Fi scanners $\mathcal{S} = \{s_1, \ldots, s_n\}$ in a public space where crowds of people are expected. These scanners could be either purposefully built Wi-Fi sniffers, access points, or any other apparatus which can pick up Wi-Fi signals in their vicinity. Ideally, scanners are positioned in such a way that they have nonoverlapping ranges, to prevent them from sensing the same signals at the same time.

People carrying Wi-Fi enabled devices pass through the public space. Their devices regularly broadcast management frames called probe requests to search for available Wi-Fi networks. Probe requests are sent out in the clear and contain, along other information, the MAC address of the sender, $a \in \mathcal{A}$ where $\mathcal{A} \subset \{0,1\}^{48}$, hereby acting as a unique identifier for the broadcasting device. Whenever such a device is identified by a scanner as passing through its range (i.e. a probe request is received from it), the scanner learns its MAC address a as well as the timestamp of reception t, and it associates it with an epoch $e \in \mathcal{E}$ such that $t_{start}(e) \leq t < t_{end}(e)$, where t_{start} and t_{end} mark the beginning and the end of an epoch and $\mathcal{E}$ denotes the set of all such epochs. We call the 3-tuple (a, s, e) a *detection*, signifying that a device with MAC address a was detected by scanner s during epoch e.

We model a crowd as a set of detections $D_{s,e}$ containing the devices detected by a scanner s during an epoch e. This decision implies two effects. First, using a set ensures that a device is counted by a scanner only once per epoch even if it may broadcast multiple probe requests. Second, we consider the number of detected devices as the number of people, this being the only information such a system can gather. We are aware that the actual number of people may be different (e.g., due to some people not carrying mobile devices) and we assume that a correction factor (e.g., how it is proposed in works such as [14]) will be applied afterwards.

2.2. Statistical counts for pedestrian dynamics

Detections made by Wi-Fi scanners can be used to derive numerous statistics on crowds. In particular, for pedestrian dynamics we are looking at two main situations in our article:

1. The crowd of people present in one place in a particular period of time, known as *footfall* (Fig. 1(a))
2. The *crowd flow* of people traveling from one place to another (Fig. 1(b))

We formally define these situations in terms of counts below.

Definition 1. Let $D_{s,e}$ be the set of detections made by a scanner s during an epoch e within a crowd-monitoring system. We define the **footfall** in the area covered by the range of scanner s during epoch e as the count obtained by computing $|D_{s,e}|$.

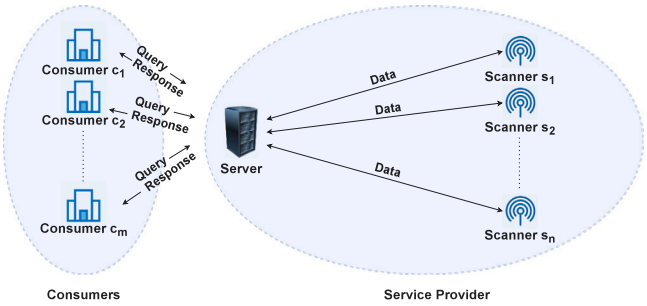


Fig. 2. Service model.

Definition 2. For a collection of sets $\{D_{s_1,e_1}, \ldots, D_{s_n,e_n}\}$ representing detections made by scanners $s_1, \ldots, s_n$ during epochs $e_1, \ldots, e_n$ within a crowd-monitoring system, we define the **crowd flow** as the intersection $\bigcap_{i=1}^{n} D_{s_i,e_i}$ over that collection. The size of the crowd flow following the corresponding path is the count obtained by computing $|\bigcap_{i=1}^{n} D_{s_i,e_i}|$.

A Wi-Fi-based crowd-monitoring system concerned with pedestrian dynamics should use readings made by scanners to produce the necessary data such that these two types of statistical counts can be computed.

2.3. Service model

Let us now model, from an architectural point of view, the actors taking part in the process. We separate data gathering and processing from its usage and propose two classes of entities (see Fig. 2).

Service Provider (SP). This is the entity which owns the sensing infrastructure and provides the crowd-monitoring service. The service is provided in the form of responses to queries received from parties interested in pedestrian dynamics insights. The responses should contain sufficient information to allow the computation of statistical counts corresponding to the situations indicated by the queries. Queries can be numerous and they can span across multiple scanners and epochs, so we assume that a separate service, acting as a central manager, assembles the data generated by scanners into responses. This service can be implemented, e.g., by a single or multiple cloud-based servers. For coherence, we are going to use the term *server* throughout the article.

Consumers. These are public or private parties interested in understanding pedestrian dynamics. They are external to the crowd-monitoring system, launching queries whenever they want to discover insights. Also, they have to process received responses in order to find out the desired statistical counts.

2.4. Security requirements

Privacy-sensitive data related to individuals is handled throughout the crowd-monitoring process. This includes unique identifiers, places and times of detections. Improper handling can lead to infringing the privacy of individuals. Therefore, we impose a number of security requirements to be fulfilled while still being able to compute statistical counts.

Consumers. The only information we allow consumers to learn is that of statistical counts as answers to queries they launch, namely the count of all detections made in either footfall or crowd flow situations. Additionally, consumers need to have a publicly verifiable identity (e.g., a public key certified by a trusted certificate authority).

Scanners. We demand that scanners are tamper-proof, such that the system can put trust in the outputs they produce. Nevertheless, even if their outputs can be trusted, we do not allow scanners to be in possession of data other than what they can generate themselves through sensing. We do allow them, though, to be aware of the consumers enrolled in the system, as they might need to generate specific data for each consumer. Still, scanners should be programmed to accept only certified consumers. Finally, at the end of each epoch scanners should discard all the detections made in that epoch, thus keeping data in clear as short as possible.

Oblivious server. The server should not be able to assemble any meaningful information from the data it handles, neither MAC addresses collected by scanners nor statistical counts nor other intermediary information related to individuals. Fulfilling this requirement protects individuals from honest-but-curious SPs, as well as in the case of an external attack on the server. Following the same honest-but-curious model, despite the server trying to extract as much information as possible from the data it handles, yet we trust it to run the protocol correctly. So we allow it to know of involved consumers, scanners and queries, as it needs to manage incoming queries as well as obliviously assembling their responses.

Non-colluding entities. We do not allow the SP to collude with any of the enrolled consumers. This means that the SP and consumers cannot cooperate outside the protocol to derive information in addition to what they are allowed to know according to the protocol. It also means that the SP cannot enroll itself as a consumer (situation prevented, nevertheless, by a previous requirement as consumers must have publicly verifiable identities).

3. Our construction

As an instantiation of the previously introduced system model, we present our construction supporting statistical counts for pedestrian dynamics as a service while fulfilling the proposed security requirements.

Each epoch, scanners collect detections and write them in detection sets. In the case of a footfall query, a scanner can simply calculate itself the cardinality of such a detection set, delivering it as a response to the server which forwards it to the intended consumer. Then, the scanner can immediately discard the data used for the calculation, as demanded by our design choice. The problem gets complicated when queries regarding crowd flows are launched. The answer to a crowd flow query is represented by the cardinality of an intersection of sets coming from multiple scanners and epochs. As detection sets are discarded, by design, at the end of each epoch, we are faced with the challenge of performing an intersection of sets without having the original sets any longer. In consequence, to support this we should come up with structures resembling sets and allowing intersections, but without knowing what is stored in the structures themselves.

3.1. Bloom filters

A Bloom filter (BF) [15] is a space-efficient probabilistic data structure typically used for storing sets of elements and allowing set membership testing. It consists of an array of m bits initially set to 0, along with k different hash functions. Whenever a set element e has to be added in the BF, the k hash functions are computed on e, each result pointing to one of the m array positions; these positions are set to 1. Correspondingly, to check whether an element is a member of a set, one has to verify if all the positions indicated by the k hash functions are set to 1. From this it immediately follows that by multiplying the bits found on the same positions of BFs which were previously encoded using the same hash functions, a new BF is obtained which approximately corresponds to the intersection of the underlying sets.

False negatives cannot occur with BFs, but false positives can, since positions corresponding to an element can be set to 1 by hashes of elements other that the one being tested for. Nevertheless, the parameters of the BF can be tuned in such a way that a desired probability of false positives p is achieved when knowing that a set of n elements must be accommodated [16].

We consider to use BFs in our system to support the result computation for footfall and crowd flow queries as the cardinality of the underlying sets. According to Swamidass and Baldi [17], the cardinality c of a BF having t bits set to 1 can be estimated as:

$$c = -\frac{m}{k} \ln\left(1 - \frac{t}{m}\right) \tag{1}$$

Service provision could thus happen in the following way. Scanners encode detections they make into BFs and transfer them, at the end of each epoch, to the server. When the server receives a query from a consumer, it starts assembling an answer by gathering the necessary data generated by scanners. In case of a footfall query, it uses that single BF of interest, while for a crowd flow query it generates a new BF by performing a bitwise multiplication of the corresponding BFs. Afterwards it shuffles the positions to remove any meaning,[1] transforming the BF into a random array of 0's and 1's, and delivers the result to the consumer. We can trust the server to do the shuffling as this is part of the protocol and it is assumed to correctly follow it. The consumer, being provided with the values of m and k, computes the desired statistical count by applying Eq. (1), as the number of 1's is not affected by the shuffling.

Until now we have a system that can address both footfall and crowd flow queries as specified by the service model. At the same time, security requirements are met at scanner and consumer level. However, the solution is not complete, the server not being yet compliant with our requirements. The MAC address space is easily enumerable [18]. Because of this, an entity (including the server) knowing the hash functions can do an exhaustive search on a BF in limited time, revealing with high probability the MAC addresses stored in it. Moreover, the server can apply itself Eq. (1) and find out any statistical counts it desires, since it has access to BFs in the clear and it sees how many 1's are in each of them.

To prevent the server from doing the previously mentioned actions, we need to combine BFs with an encryption scheme such that the server would handle only encrypted data that it cannot decrypt. To satisfy our security requirements while maintaining the functionality unchanged, such an encryption scheme should have the following properties:

- Encrypting the same value on several occasions should produce different ciphertexts. Otherwise, as BFs contain only 0's and 1's, an attacker could learn, despite encryption, the positions containing identical values, thus being able to infer how the original BFs looked like.
- It should allow multiplications under encryption, such that the decryption of the result of a multiplication of ciphertexts equals the result of the multiplication of the unencrypted values behind those ciphertexts.
- It should be a public-key encryption scheme such that everyone (especially the scanners) can encrypt values but that only the consumer, who is the only one having the corresponding secret key, can decrypt it.

3.2. Homomorphic encryption

Homomorphic encryption (HE) [19] is a type of encryption that allows mathematical operations to be performed directly on encrypted data without requiring decryption. The results of such operations, encrypted as well, are the same as if the operations were performed on the unencrypted data. HE includes different classes of encryption schemes depending on how many types of operations they allow and how many times they can be applied. Partially homomorphic encryption (PHE) is a class of schemes that allow performing a single type of operation under encryption, either addition or multiplication, for an unlimited number of times. PHE sufficiently satisfies our requirements, having specified in 3.1 that we are looking for an encryption scheme that allows multiplications under encryption (i.e. a single type of operation).

ElGamal encryption [20] is such a PHE cryptosystem which allows multiplications under encryption; we are going to use it in our construction. The algorithm is asymmetric, using a public key for encryption and a private key for decryption. Furthermore, the algorithm is probabilistic, involving randomness in the encryption process, so that encrypting the same value several times yields different and indistinguishable ciphertexts.

Combining BFs with HE closes the circle of our system model. The complete service provision happens as follows. When a consumer enrolls in the system, it generates a public–private key pair and it gives the public key to the SP, which distributes it to its scanners. At the end of an epoch, scanners write detections into a BF. For each enrolled consumer, they make a copy of that BF and encrypt each position with the public key of that specific consumer. We note that 0's are represented as random numbers as ElGamal can deal only with positive integers. They then send the resulting encrypted BFs (EBFs) to the server and discard the original detections. When the server receives a query from a consumer, it acts in the same way as before, just that this time it obliviously handles encrypted data and, if necessary, it performs bitwise multiplications under encryption, as depicted in Fig. 3. To estimate the statistical count from a response, a consumer iterates through it, decrypts the ciphertexts, sums up the 1's to find out t and applies Eq. (1).

4. Accuracy analysis

In this section we explore the potential of our solution to estimate statistical counts for pedestrian dynamics. We generate detections[2] emulating footfall and crowd flow situations for a whole range of sizes. We feed these detections as inputs to our system and perform an accuracy analysis of the responses, i.e. comparing the statistical counts generated through our system with the actual counts of the emulated situations. In doing this, we examine different combinations of parameters for BFs, as this is the part in our solution which influences the statistical counts. For the hashing part, we choose to use MurmurHash3 [21] with different seeds, a fast hash function which despite its non-cryptographic nature is suitable for our case where the positions written in BFs are not available in the clear.

We expect to see differences between what is estimated and ground-truth values because of two main reasons. First of all, the result of the estimation formula is the number of elements having the *highest likelihood of being members*, given the state of the BF; the actual number can be different depending on the probabilistic properties of the underlying set. Secondly, *false positives* can be encountered when working with BFs, which means that the k positions corresponding to an element tested as present could have been set to 1 by the hash values of other elements; this can also lead to differences in counts when intersecting BFs for crowd flows.

The accuracy metric that we propose measures the closeness of the estimated count c to the real count c_t and is formally represented below. Please note that we choose to interpret estimated counts that are more than twice as high as the real counts as having accuracy 0 since they are more than 100% off from the real counts.

$$Acc = max\left(1 - \frac{|c - c_t|}{c_t}, 0\right) \tag{2}$$

[1] Meaning is lost as shuffling is done randomly. Randomness is based on a distinctive runtime-dependent value that is not memorized. Note though, even if it had been memorized, reaching a consumer would have demanded collusion, which is explicitly prevented by the security requirements.

[2] To have precise control over the detection sets, we choose not to do a physical level simulation, but generate the detections directly, as if they were detection data collected a priori.

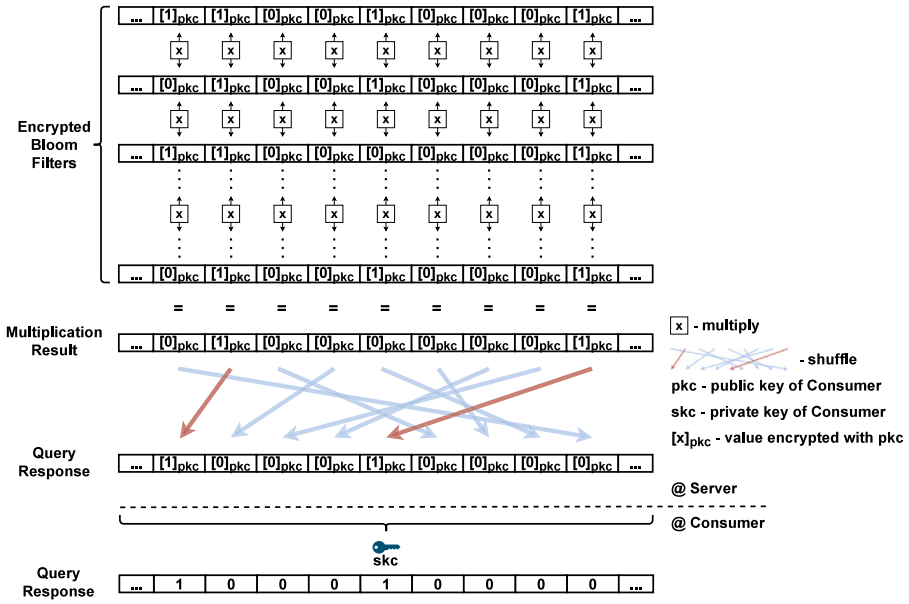


Fig. 3. Preparing response to a crowd flow query by performing multiplications under encryption and shuffling. Consumer decrypts response, then counts the 1's and estimates the statistical count using Eq. (1).

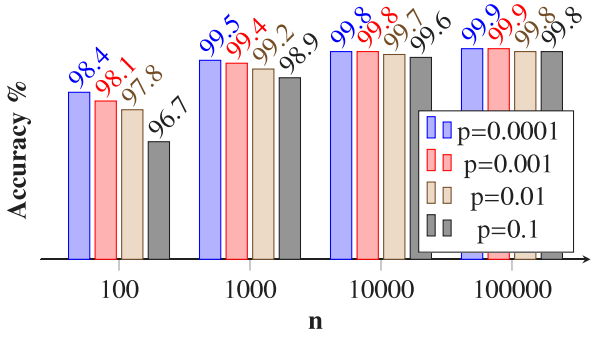


Fig. 4. Worst-case accuracy for footfall queries when dealing with different values of n and p.

Table 1
BF parameters.

p ↓ n →	100	1000	10 000	100 000	k ↓
0.0001	m = 1918	m = 19 171	m = 191 702	m = 1 917 012	13
0.001	m = 1438	m = 14 378	m = 143 776	m = 1437 759	10
0.01	m = 959	m = 9586	m = 95 851	m = 958 506	7
0.1	m = 480	m = 4793	m = 47 926	m = 479 253	3

BF parameters can be set in such a way that a desired false positive probability p is obtained when having a set containing n elements. The length m of the BF can be computed as $-n \ln p/(\ln 2)^2$ and the optimal number of hash functions k as $-\log_2 p$. In our crowd-monitoring setting, choosing values for n and p implies accommodating a maximum number of devices n detected by a scanner during an epoch while allowing a maximum false positive probability p for the resulting BFs. In Table 1 we display the setup parameters that we use throughout the rest of the article.

4.1. Accuracy of footfall queries

In principle, a lower m would be desirable for performance reasons, as it would mean less cryptographic operations to be performed. However, reaching a lower m requires opting for a higher probability of false positives p, which may have consequences on the accuracy of queries. To get a clear understanding of the link between the choice of p and the accuracy of footfall queries, for fixed values of n, we run experiments ranging p, as displayed in Table 1. As identifiers, we generate random MAC addresses coming from a uniform distribution, resembling, thus, real-world deployments, where a cryptographic hash is usually applied on the real identifiers before being processed; we will use the same way of generating addresses throughout the rest of this section.

We plot in Fig. 4, as worst case, the minimum mean accuracy measured for each choice of n and p when handling between 0 and n devices, with a step equal to 10% of n and doing 100 runs per step with different addresses each run. Results show that indeed lower accuracy is to be expected for footfall queries when choosing a higher p, a trend which is consistent across all the tested values of n. However, the impact is not significant, as even for the highest tested value of p, the accuracy does not get below 96.7%, 98.9%, 99.6% and, respectively, 99.8% for the 4 different values of n. For example, a worst-case accuracy of 98.9%, as when n is 1000 and p is 0.1, comes from estimating 784 instead of 800 devices.

By looking at the formula for estimating statistical counts (Eq. (1)), when BF parameters are fixed, we see that the estimation and hence the accuracy of the concerned footfall query depend only on the number of bits t set to 1. In our case, t is dictated by the number of sensed devices c_t. Let us see, thus, what accuracy we can expect from footfall queries when *ranging* c_t. To see the trend, we run an experiment in which we fix n to 1000 and p to 0.01 and range c_t, starting from 0, going to n and then even beyond, up until it leads to a t getting close to m. We increase c_t with a step of 100, we do 100 runs with different addresses for each step and plot mean accuracies together with standard deviations and confidence intervals in Fig. 5.

The accuracy of the statistical counts stays above 99.2% when $c_t \leq n$, as also presented in Fig. 4, showing a standard deviation of 0.04% in that worst case. For this experiment, the threshold for which c_t leads to a full BF is around 10 000 devices. Regarding footfall only, the accuracy of the statistical counts stays above 95.3% even when larger-than-designed-for crowds of up to the mentioned threshold arrive, though with a higher standard deviation of 4.5%. We have run experiments with other values of n and p as well and we confirm that the same trend, with accuracies in the range of 90%'s, is to be seen. Please note, however, that in cases when the crowd is inflated beyond the designed maximum, the false positive probability inflates as well, which deems the results as problematic for further counting crowd flows.

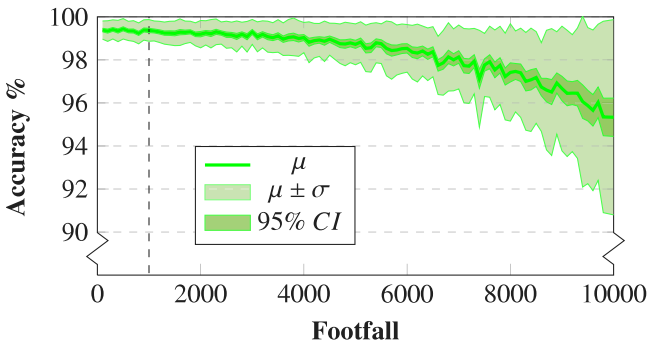


Fig. 5. Accuracy of footfall queries when increasing the number of detected devices until completely filling up the BF. Parameters: $n = 1000$ and $p = 0.01$. The vertical dashed line marks n.

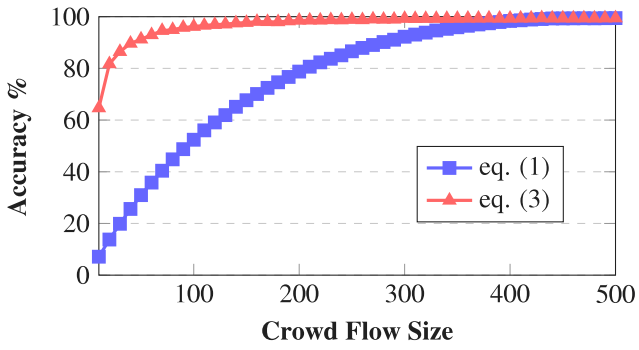


Fig. 6. Preliminary experiment with $n = 1000$, $p = 0.01$, crowd size 500 in both locations, crowd flow size ranges between 10 and 500.

4.2. Accuracy of crowd flow queries

A crowd flow is represented in data as a BF resulting from performing a bitwise multiplication of other BFs. This result is an approximation of the intersection of the underlying sets, as the probability of false positives leads to having false matches in an intersection. We say that the BF resulting from the bitwise multiplication of BFs which represent sets of detections is different from the BF representing the intersection of those sets of detections. It tends to be more different when more false matches occur. For this reason, in the same manner, the statistical counts estimated by Eq. (1) will also get farther from the actual counts. This aspect was also noticed by Papapetrou et al. in [22]. They propose to improve the estimation by taking into account, besides the 1's in the resulting BF, the 1's in the BFs used in the bitwise multiplication. For t_1, t_2 and $t_\wedge$ as the number of bits set to 1 in two BFs to multiply and, respectively, in the result, the estimation formula is:

$$c_\wedge = \frac{\ln\left(m - \frac{t_\wedge \times m - t_1 \times t_2}{m - t_1 - t_2 + t_\wedge}\right) - \ln(m)}{k \times \ln\left(1 - \frac{1}{m}\right)} \tag{3}$$

We run a preliminary experiment in which we fix n to 1000 and p to 0.01. We intersect two crowds of 500 people each and range the crowd flow size between 10 and 500 people. We plot the accuracy of statistical counts while using both estimation equations to understand the dimension of the improvement. In Fig. 6 we see that the improvement is sensible, the accuracy much faster approaching 100% when using Eq. (3) instead of Eq. (1).

For applying Eq. (3), a consumer would need to know, along the crowd flow query response, the answers to the associated footfall queries in order to determine t_1 and t_2. This is not in contradiction with our system model, as consumers are allowed to launch such queries.

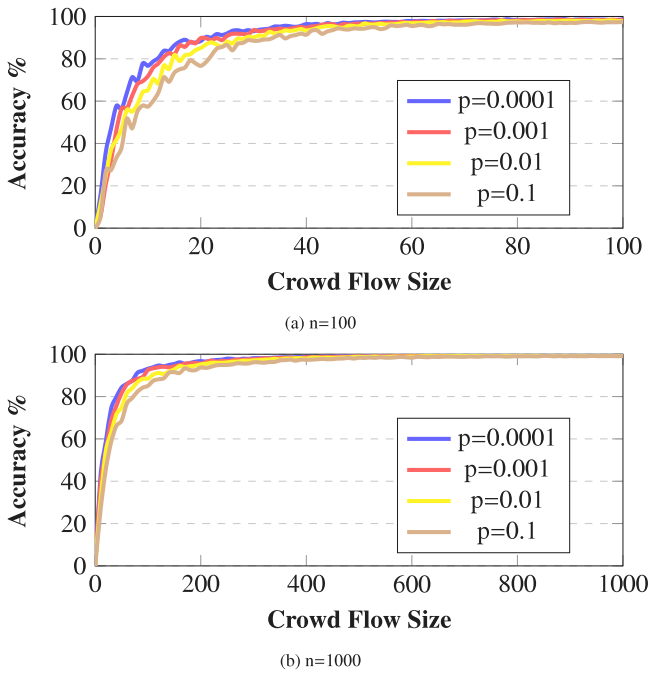


Fig. 7. Accuracy of crowd flow queries for n first fixed to 100 and then to 1000, for different values of p, when ranging the crowd flow size between 0 and n. Plots display worst-case scenario, crowds at the ends of the crowd flow being fixed to the maximum value (i.e. n).

Also, it does not imply any additional computationally expensive operations. Hence, we use this alternative in the rest of the evaluation for estimating statistical counts on crowd flows.

As we have previously mentioned, the probability of false positives p is an important parameter when it comes to estimating statistical counts on crowd flows. Inserting elements in a BF with higher p leads to a higher density of 1's than by inserting the same elements in another BF with lower p; consequently, this leads to more false matches when using the former type for estimating crowd flows. To assess how big of a problem this is, we propose the following experiment. For a fixed n, resembling a worst-case scenario, we take two crowds of size n (i.e. the maximum accommodated number of devices) and range the size of the crowd flow happening between them from 1% to 100% of n. We plot the mean accuracy (from 100 runs) of the estimated crowd flows when choosing p to be 0.0001, 0.001, 0.01 and 0.1. We run the experiment four times, for n equals 100, 1000, 10 000 and 100 000. We display the results for the first two values of n in Fig. 7.

The accuracy of statistical counts on crowd flows shows logarithmic growth for all the setups we have tested, from low when the crowd flow is small in comparison with the intersected crowds to high for more steady crowd flows. Also, as expected, higher p means constantly having lower accuracy, but not significantly lower; e.g., for n equals 10 000 and 100 000, lines representing the accuracy for different p-s almost entirely overlap, this being the reason for not showing their graphs. This is important, as we recall that p inversely influences the length m of BFs (see Table 1) and, thus, the performance of the system, a phenomenon which we will study in detail in the upcoming section. Another pattern we observe is that for setups where the maximum crowd size is higher, the accuracy gets close to 1 quicker. To illustrate this better, we draw another graph (Fig. 8) based on the same experiment, this time showing, for a fixed value of p and all the four values of n, the minimum crowd flow size as a percentage of n for which accuracies of at least 50%, 60%, 70%, 80% and 90% are reached. To reach, for example, an accuracy of at least 90%, the crowd flow size should be at least 29% of the initial crowds when n is 100, 10.8% when

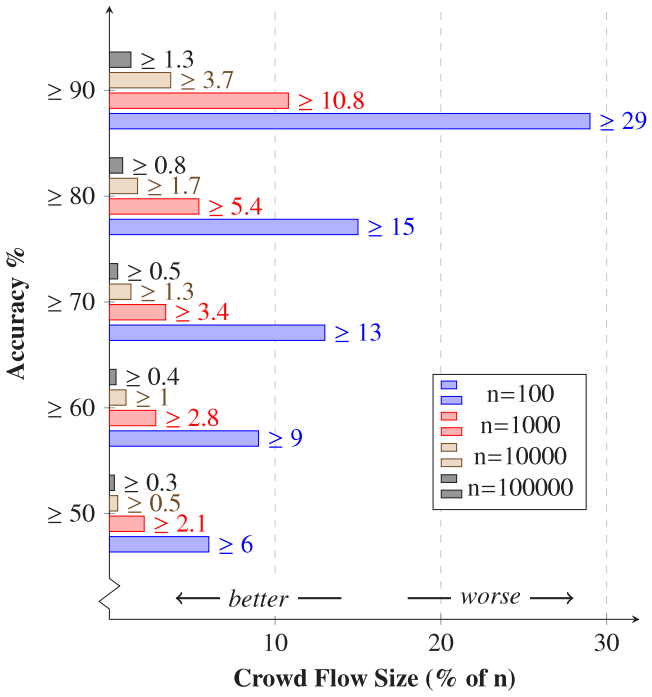


Fig. 8. Fixing p to 0.01 and initial crowds as worst-case to maximum (i.e. n), we display the crowd flow size as a percentage of n for which accuracies of at least 50%, 60%, 70%, 80% and 90% are reached.

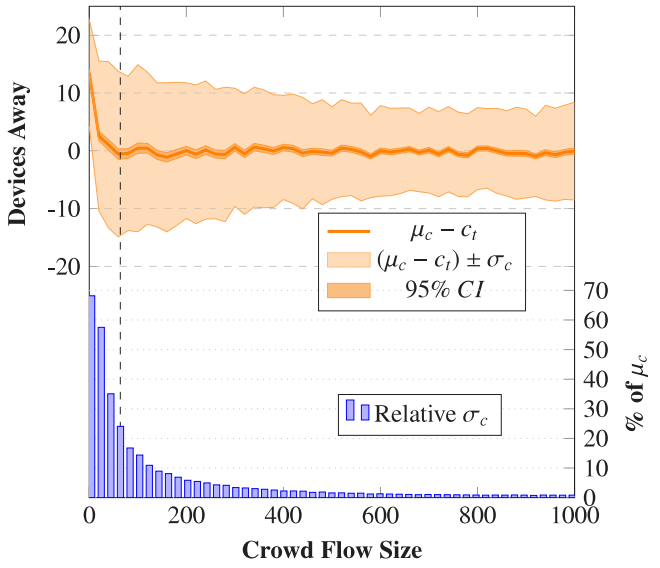


Fig. 9. We fix p to 0.01, n to 1000, initial crowds as worst-case to maximum (i.e. 1000) and range crowd flow size between 0 and 1000. In the upper part we display mean estimated counts as *devices away* from real counts, together with standard deviations of the estimated counts. Below we plot standard deviations as percentages of the mean estimated counts. The vertical dashed line marks the minimum crowd flow size for which estimations always turn positive.

n is 1000, 3.7% when n is 10 000 and 1.3% when n is 100 000. We remind, though, that these are results for worst-case scenarios, when initial crowds are equal to n; when initial crowds are smaller, accuracy thresholds are reached even quicker.

As seen from Fig. 7, a low accuracy is to be expected for crowd flows that are small in comparison with the intersected crowds, but a low accuracy does not necessarily mean something bad for crowd-monitoring purposes, especially when talking about small numbers. For example, having two crowds of 500 people with an actual crowd flow of 20 people between them and an estimated count of 15 or 25 incurs an accuracy of 75%, which might seem low at first sight. In reality, the estimation is only 5 devices away from the real count, which can be very well considered an insignificant error given the size of the initial crowds, i.e. being 100 times bigger. This leads us to looking into the actual distances between the estimations and the real counts, to get better insights for such situations where the accuracy does not tell too much. For this, we run an experiment in which we fix p to 0.01, n to 1000, initial crowds to n and range the size of the crowd flow between 0 and 100 with a step of 20. We do 1000 runs per step, each time using different addresses for devices. We show the results in Fig. 9. For the upper part of the graph, we compute the mean estimated count μ_c, as well as the standard deviation σ_c. We plot the difference between μ_c and the real count c_t, along with σ_c and confidence interval, to see how far away from the real counts, corresponding to 0 on the y axis, and in which direction the estimations are. In the lower part of the graph we plot σ_c as a percentage of μ_c, a measure commonly known as *relative standard deviation*.

The maximum standard deviation is 14.32 when the real count is 40 devices, a point beyond which the mean estimation stabilizes as almost identical to the real count. It decreases to the right, the estimations getting closer to the mean. It also decreases to the left in a counterintuitive manner, due to a positive estimator bias inflicted by estimations which are negative, according to the formula, but we set them to 0, as statistical counts cannot be negative. The relative standard deviation quickly decreases, as expected from previous experiments concerning accuracy, the estimations getting closer to the real counts as the crowd flow size increases. Considering the case of the maximum standard deviation, 68% of the estimations of a crowd flow of 40 devices traveling between two crowds of 1000 devices fall between 26.63 and 55.27, with a mean of 40.95 devices. The minimum encountered standard deviation is 6.78 for a real count of 720 devices; for this case, 68% of the estimations fall between 714.21 and 727.77, with a mean of 720.99.

5. Implementation & performance analysis

To understand the cost dimensions our solution incurs at different levels of the system and bearing in mind that adding homomorphic encryption may generate considerable overhead, in this section we carry out an actual implementation. We describe the necessary algorithms and we deploy them at scanner, server and consumer. Based on this implementation, we conduct a performance analysis addressing several relevant and practical concerns. We use again MurmurHash3 with different seeds as a hash function for BFs. We instantiate ElGamal using the NIST P-256 elliptic curve [23] and we use the SCAPI[3] library [24] for homomorphic encryption support.

5.1. Scanner-side

Scanners execute Algorithm 1, which takes each detection made in an epoch, applies the k hash functions on it, sets the corresponding BF positions to 1 and then it applies the ElGamal encryption. They run this algorithm for each consumer enrolled in the system.

We implement this on a Raspberry PI 4B, which uses a Broadcom BCM2711 SoC, with a 1.5 GHz 64-bit quad-core ARM v8 Cortex-A72 processor, 8 GB of DDR4 RAM memory, 16 GB microSD memory card and it is running Ubuntu 20.10 as OS. We implement both serial and parallel versions of the algorithm, using C++11 threads for parallelization.

[3] https://github.com/cryptobiu/libscapi.

```
Input: pkc //Public key of
       consumer;
Input: DSE[ ] //Detections;
Input: m //BF length;
Input: k //Number of hash
       functions;
Input: H[k] //The hash functions;
Output: EBF[ ] //Encrypted BF;
BF := [0];
/* Set BF positions corresponding
   to detections */
foreach DSE as currentMAC do
    for i := 0 to k − 1 do
        pos := H[i](currentMAC);
        BF[pos] := 1;
    end
end
/* Encrypt each BF position */
for i := 0 to m − 1 do
    if BF[i] = 1 then
        EBF[i] :=
          ElGama-lEnc(pkc,BF[i]);
    else
        EBF[i] :=
          ElGamalEnc(pkc,rand());
    end
end
return EBF;
```

Algorithm 1: Algorithm running on a scanner, for each enrolled consumer, at the end of an epoch.

Supposing that a single consumer is enrolled in the system, we want to see how long it takes for a scanner to process the readings in an epoch. Analyzing the algorithm, for $d = |DSE|$, we see that for an epoch a scanner must compute $d \times k$ hashes and m ElGamal encryptions. In Fig. 10 we display the results of an experiment timing such executions for serial and parallel implementations. In the former subfigure, we show the timings when p influences the run time, n being fixed to 1000 (axis y linear), while in the latter we fix p to 0.01 and range n (axis y logarithmic). Hash operations, in comparison with encryptions, are negligible, their duration falling in the range of nanoseconds, while encryptions are in the milliseconds range. This is why the values in Fig. 10 are mostly dictated by the resulting m for given p and n. For a crowd of maximum 1000 people, readings can be processed by the chosen scanner in less than 93 s for any tested value of p in the serial implementation and in less than 25 s for the parallel implementation; when $p = 0.1$, the parallel run time is as low as 6 s. Larger crowds can be processed too in a reasonable amount of time judging by the parallel timings, such as approximately 2 min for a crowd of 10 000 and 20 min for a crowd of 100 000, the maximum size we have tested for.

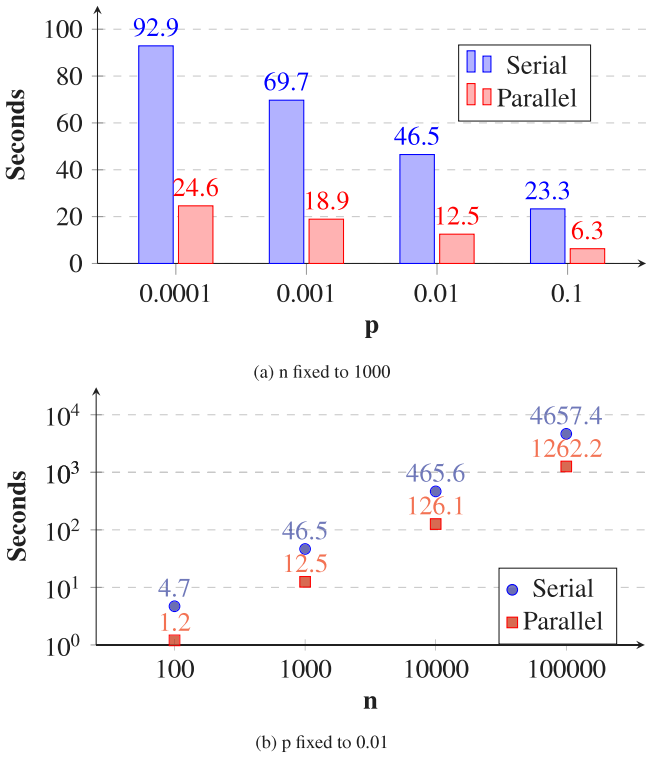
To keep up with the incoming detections, a scanner must process the readings for all the consumers enrolled in the system in a time period which is less than the epoch length. Therefore, for a scanner achieving an average hashing duration in parallel t_h, an average encryption duration in parallel t_e, for the system parameter epoch length e_l and BF parameters k, n and m, the maximum number of consumers enrolled in the system at the same time can be computed as:

$$n_c = \left\lfloor \frac{e_l - k \times n \times t_h}{m \times t_e} \right\rfloor \tag{4}$$

In particular, for the resource-constrained device which we used as a scanner and a fixed epoch length of 5 min, we plot n_c in Fig. 11 when ranging n for different values of p. More consumers can be supported for higher values of p. For example, when n is 10 000, n_c is 1, 1, 2 and 4 for p equals 0.0001, 0.001, 0.01 and 0.1. The x axis extends up until no more consumers can be supported for the lowest value of p, which is when n reaches 11 614. For the highest value of p, at least one consumer can be accommodated up until n reaches 46 455, while for crowds of 1000 people, no less than 46 consumers can be satisfied.

5.2. Server-side

A server gathers and stores EBFs from its scanners. Whenever it receives a query from one of the enrolled consumers, it runs Algorithm 2. Essentially, it multiplies under encryption, position-wise, the EBFs corresponding to the scanners and epochs concerned by the query and then shuffles the results.



(a) n fixed to 1000

(b) p fixed to 0.01

Fig. 10. Time needed for processing the readings in an epoch for a single consumer.

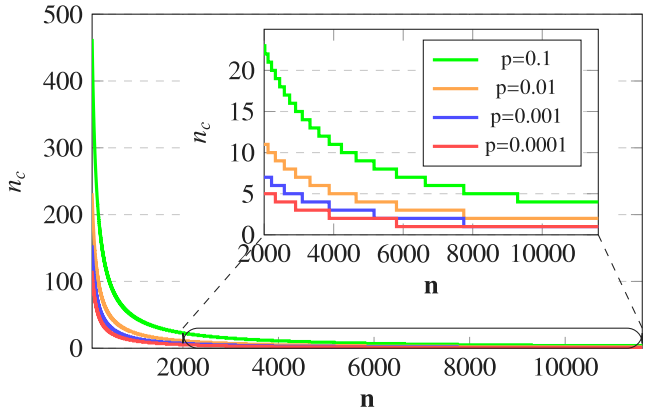


Fig. 11. The number of consumers n_c enrolled in the system which can be supported by a scanner, when the epoch length is fixed to 5 min.

For a query of complexity q representing the number of EBFs to combine, a server must do $m \times q$ homomorphic multiplications under encryption, plus a final shuffle operation. We plot in Fig. 12(a) the computational effort needed to address queries of complexities between 1 and 10 for different values of p, where we consider a homomorphic multiplication to take one computational unit and the shuffle operation as negligible.

We implement Algorithm 2 both on a laptop and a more powerful cloud server to see how quickly different servers can provide responses to queries launched by consumers. We use the same ElGamal configuration as for the scanners. The algorithm is embarrassingly parallel, the iterations through the i loop showing no interdependencies, so

```
Input: pkc //Public key of consumer;
Input: EBFs[ ][ ] //Encrypted BFs to combine;
Input: q //EBFs length;
Input: m //BF length;
Output: QRes[ ] //Query response for consumer;

EBFRes := [ ];
for i := 0 to m − 1 do
    mul := EBFs[0][i];
    for j := 1 to q − 1 do
        currentEBF[] := EBFs[j];
        mul := ElGamalMul(mul,currentEBF[i]);
    end
    EBFRes[i] := mul;
end
/* Rearrange positions in random order */
QRes := shuffle(EBFres);
return QRes;
```

Algorithm 2: Server assembling query response for a consumer.

we parallelize it using C++ 11 threads. The laptop is running Ubuntu 20.04 x86_64, having 8 GB RAM and a 4-core Intel(R) Core(TM) i5-10210 CPU @ 1.60 GHz, while the cloud server is running Ubuntu 18.04 x86_64, it has 16 GB RAM and a 16-core Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz. In Fig. 12(b) we plot the mean query response time in seconds from 10 runs on the two server configurations, as well as minimum and maximum response times, when q ranges between 1 and 10 and BF parameters are fixed. The responses are almost instant for footfall queries, only shuffling being necessary. The response time increases with q as expected from Fig. 12(a), remaining below 10 s for the most complex query launched on the cloud server configuration, while even the basic laptop could provide responses in comparable times. Most importantly, almost ideal speedup can be easily achieved in a cloud environment by adding additional cores next to the existing ones, as the most computationally expensive parts of the algorithm are executed in parallel.

Besides response time, we also evaluate what throughput can be achieved by the two server configurations. We first show in Fig. 13(a) the throughput in *homomorphic multiplications under encryption per minute* (HMs/min), which is independent of BF-related parameters and independent of the complexity of launched queries. Having this information, a SP can form an idea on how to choose the parameters of the system in harmony with the expected crowd sizes, the accuracy desired from the system, the number and complexity of the queries expected from consumers. To exemplify, in Fig. 13(b) we then evaluate the throughput in *queries per minute* achievable by a system which must accommodate crowds of up to 1000 people, with a probability of false positives 0.01. We display on the graph only results for query complexities q between 2 and 10, as for footfall queries (i.e. $q = 1$) no additional homomorphic operations have to be done. In other words, for such setup parameters, a server can address approximately 35 000 footfall queries per minute or tens of crowd flow queries per minute, with numbers decreasing when queries' complexity rises.

5.3. Consumer-side

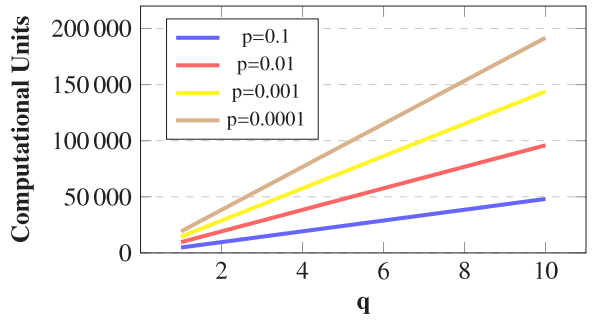
The statistical count related to a query is estimated by the consumer based on the response received from the server in Algorithm 3.

The computational load is approximately equal to that of a server assembling a response for a query of complexity 1, as the ElGamal multiplication and decryption operations have similar duration and the rest of the operations can be considered negligible. We plot in Fig. 14 the time needed by the laptop configuration acting as a consumer, when ranging m between its minimum and maximum values in Table 1. It ranges between less than a second for the lowest and approximately 12 min for the highest value of m. Additionally, we indicate on the graph the computation time for a setup with $n = 1000$ and $p = 0.01$, which is 3.8 s. We also indicate through a dashed line the value of m for which $n = 10000$ and $p = 0.0001$; all the parameter combinations except those with $n = 100000$ find themselves to the left of this line and lead to computation times lower than 77 s. Eventually, for $n = 100000$ we explicitly indicate the corresponding computation times.

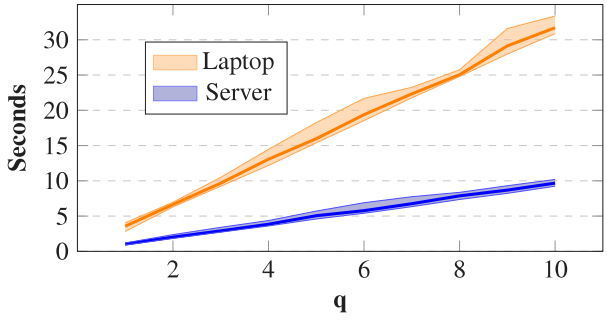
```
Input: skc //Private key of consumer;
Input: m //BF length;
Input: k //Number of hash functions;
Input: QRes[ ] //Query response;
Output: c //Estimated statistical count;

t := 0;
for i := 0 to m − 1 do
    x := ElGamalDec(skc, QRes[i]);
    if x = 1 then
        t := t+1;
    end
end
/* Estimate count */
c := -m/k×ln(1-t/m);
return c;
```

Algorithm 3: Statistical counts estimation.



(a) n=1000, showing computational units for different values of p

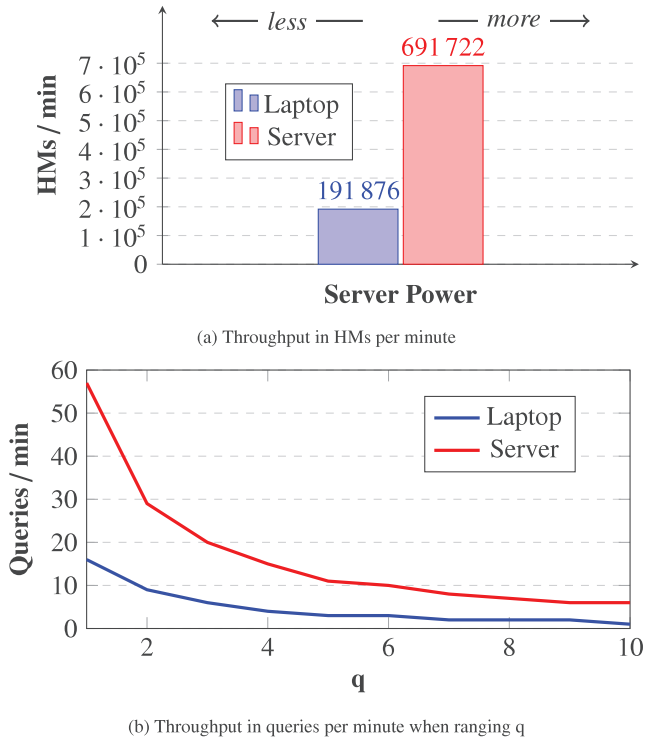


(b) n=1000, p=0.01, showing mean, min and max runtime in seconds for laptop and server configurations

Fig. 12. Computing the answer for a query on a server for query complexities between 1 and 10.

6. Real-world case study: Assen TT festival

It is important to go beyond simulations and test our system using real-world data. Such data has some particularities that can be hardly reproduced in simulations. In a real-world setting, crowd sizes and directions of flows are generated by real people and their movements. The identifiers sensed by our system are, too, real, not synthetic. The distribution of the identifiers is unique, hard to mimic, as their appearance and occurrence is dictated by the devices and movements of the aforementioned real people, and it is also influenced by the particular setup in which the sensing takes place. In other words, facing our system with real-world data, reproducing the execution of it and analyzing the system's behavior is the closest we can get to a real deployment.



(a) Throughput in HMs per minute

(b) Throughput in queries per minute when ranging q

Fig. 13. Throughput for laptop and server configurations.

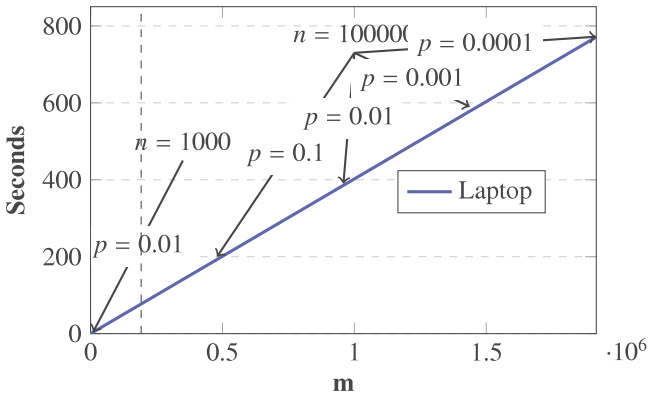


Fig. 14. Consumer computation time. Dashed line is at $m = 191\,702$; all computation times for n is 100, 1000 and 10 000 are to the left of it, regardless which value of p we choose from Table 1.

Every year, in the city of Assen, The Netherlands, a motorcycle grand prix[4] takes place which gathers crowds of people from all over Europe. In parallel, the municipality organizes the TT Festival[5] in the days before and after the race. The festival is spread across the city center, which is open only to pedestrians for the duration of the festival. There are concerts taking place on multiple stages, places for motorcycle stunts, street-food areas and amusement parks. Typically, more than a hundred thousand people visit this event, leading to a considerable amount of pedestrian movement between the attractions.

Wi-Fi data gathering from scanners installed through the city was performed during the 2015, 2016 and 2017 editions [25]. In this article, we use the dataset from 2017, when 30 Wi-Fi scanners were deployed in the city of Assen for 12 days, covering the whole period

[4] https://www.motogp.com/en/event/Netherlands.
[5] https://ttfestival.nl/.

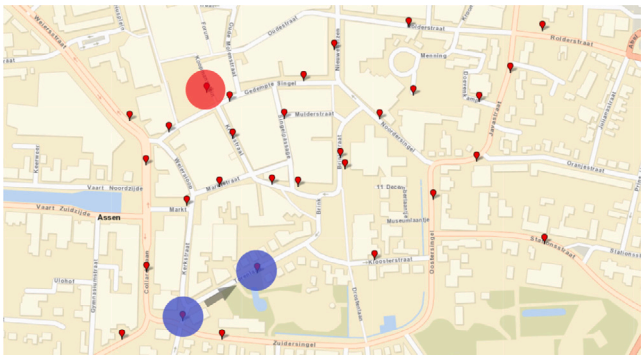


Fig. 15. Placement of scanners in the city center of Assen.

of the TT Festival as well as several days after the festival. In total, there were 26 414 742 detections of devices having 176 888 different identifiers. A map with the placement of scanners is displayed in Fig. 15. Highlighted in red is the scanner placed in Koopmansplein, corresponding to the most visited area, having 1.6 million detections throughout the whole period. Also, we highlight in blue the scanners on Torenlaan street, which find themselves on the main path leading to a big parking on the edge of the city center and having a whole range of crowd flows happening between them.

For the setup phase, our system needs the following parameters: e_l — epoch length, n — a maximum number of devices expected near a scanner within the chosen epoch time, p — the probability of false positives when n devices are present. Being interested in capturing both footfall and crowd flow situations, we choose to fix e_l to 5 min. This is a time interval long enough to ensure that we capture probe requests from most of the devices broadcasting such messages near a certain scanner within an epoch, as well as short enough to interpret a detection of the same device at another scanner in a subsequent epoch as being part of a crowd flow. Analyzing the dataset, we see that for a 5 min long epoch most of the detection sets contain less than 1000 devices, with very few only marginally exceeding this threshold, which makes 1000 an appropriate setup value for n. We set p to 0.01 as we have seen from Sections 4 and 5 that such value is expected to produce highly accurate responses and it is safely supported by the resource-constrained device that we use as scanner in our experiments.

We emulate the scanner in Koopmansplein by feeding the Raspberry Pi with detections from the dataset at the exact same pace as they happened in real life. We choose a continuous period spanning across 9 days, containing 3 festival nights followed by 6 festival-free days; this choice is consistent throughout the rest of the section, representing an interval in which all the concerned scanners (including those in Torenlaan street) uninterruptedly gathered detections. We allocate detections to corresponding epochs, according to the timestamps attached to them in the dataset, and we subject them to Algorithm 1 while using the same hash functions and encryption scheme as in the previous section. Getting as close as possible to reality, we want to test how well a scanner performs, as part of an actual implementation of our system, when faced with a real-world flow of detections. Also, we want to see how accurate the statistical counts of footfall queries are for such a setup. We are aware that based on Section 4 one can form an idea about what to expect; nevertheless, what we perform here is a neat reproduction of the festival environment as if our system was implemented there, dealing with real distributions of identifiers and real detections of them across time.

In Fig. 16 we plot the statistical counts estimated by a consumer based on the answers received from the system, the absolute errors of these counts (i.e. the absolute differences between real counts and estimations), as well as the accuracy according to the definition. The average processing time on the scanner was 12 s, with minor variations

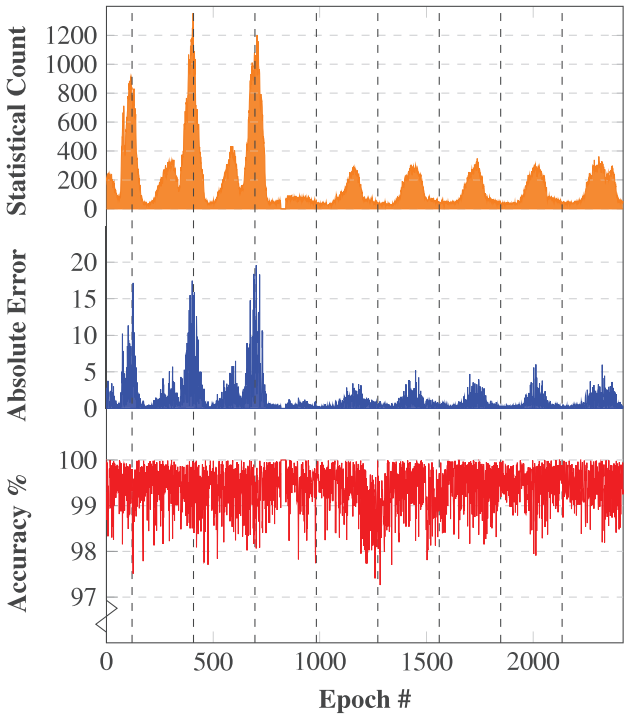


Fig. 16. Estimating footfall in Koopmansplein during festival days and afterwards. Vertical dashed lines indicate midnight.

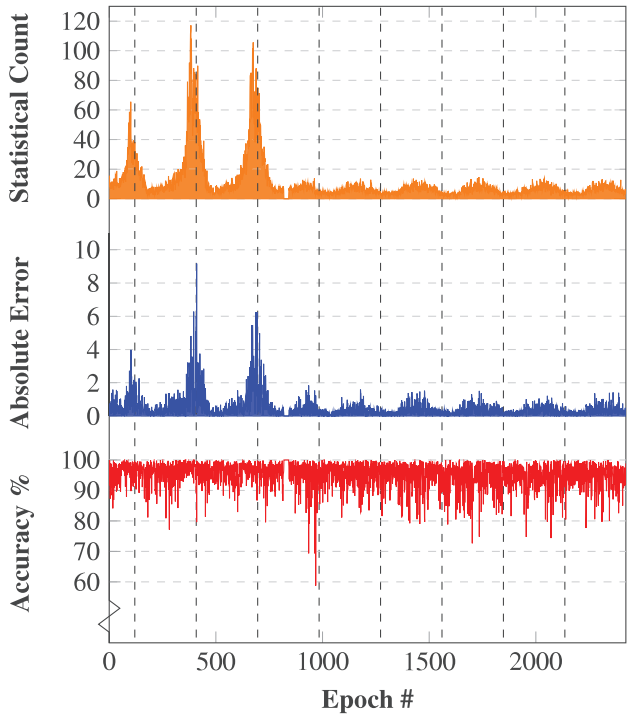


Fig. 17. Estimating crowd flow size on Torenlaan street. Vertical dashed lines indicate midnight.

depending on footfall size. The highest absolute error is 19.58, where instead of 1012 devices the estimation was 992.42. The lowest observed accuracy is 97.2%, where instead of 36 devices the estimation was 35.01. The small gap in the fourth day corresponds to a period when readings did not reach the server, which is common across different scanners.

Using the same setup parameters, in Fig. 17 we have a look at the crowd flows between the two scanners in Torenlaan, flowing in the direction of the parking, which target devices making the transition between the two places in consecutive epochs. We plot the statistical counts a consumer estimates based on the results returned by our system, together with absolute errors, as well as the obtained accuracies. Please note that there is a one-to-one match between epochs in Figs. 17 and 16. The largest crowd flow has 122 devices in it, with an estimation of 117.18 and an accuracy of 96%. The lowest observed accuracy is 58.7% for a real count of 3 and an estimation of 4.23, though for 88.5% of the 2422 crowd flows the accuracy stays above 90%. The highest observed absolute error is 9.17, which happened for a real count of 45 estimated as 54.17. However, for 98.7% of the crowd flows, the estimation is *less than 3 devices away* from the real count.

In Fig. 18 we group the crowd flows by their real count found on the x-axis. In the upper part we display the estimated counts as devices away from the real counts. For comparison, we show them overlapped with the graph from Fig. 9, which was plotting, as a worst-case scenario, the mean and standard deviation for crowd flows originating from crowds sized 1000. In the lower part we plot, for each real count, the mean accuracy of the estimated counts.

All crowd flows happening on Torenlaan street are estimated within the boundaries of one worst-case standard deviation from the real count. Small crowd flows are much closer to the real count than to the expected mean as they come from much smaller initial crowds than they do in the worst-case; for Torenlaan street, crowd flows smaller than 10 usually result from intersecting initial crowds smaller than 100, compared to 1000 in the worst-case, systematically leading to less false matches and more accurate estimations. Accuracy-wise, the mean stays

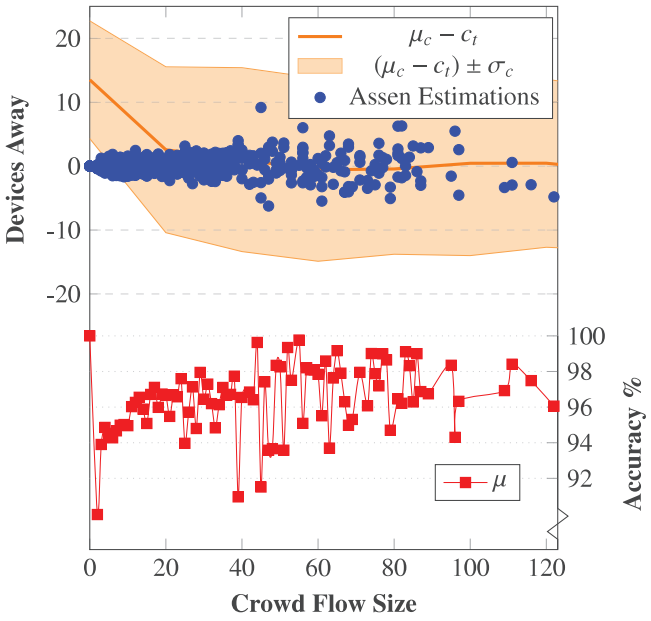


Fig. 18. Estimating crowd flow size on Torenlaan street, grouped by real count and compared with Fig. 9.

above 89.9% for all the encountered crowd flow sizes. The decrease in smoothness once the crowd flow size increases comes from having fewer samples for larger crowd flows.

7. Discussion

7.1. Clarifications on statistical counts

Wi-Fi scanners cover a certain range where they gather detections from. This range is influenced by the physical characteristics of the

installation location and *it fluctuates in time* because there are phenomena that are known to influence Wi-Fi signals propagation, such as reflection, refraction, scattering, diffraction, absorption or shadowing. So when we say that a device is near a scanner, it means that the device is in the range of the scanner at a specific moment. However, being in the range of a scanner does not guarantee that a device is detected. For a detection to happen, a device should broadcast a Wi-Fi probe request message within that epoch (sending patterns differ across devices) and that message should reach the scanner (packet loss can occur due to congestion or interference).

Literally, the footfall as we model in our system is the number of *distinct identifiers* in the Wi-Fi messages reaching a scanner within an epoch. This number is different from the number of people in that area, as there are people that carry no device while others carry more than one device. Moreover, this is even different from the number of detected devices, as there might be devices which use MAC address randomization and re-randomize their broadcasted identifier in a period of time shorter than the epoch duration. Also, it is worth mentioning that this number is independent of how many times the identifiers are seen, since our system uses *detection sets*.

Continuing along the same lines, a statistical count on a crowd flow is the number of distinct identifiers found in the Wi-Fi messages which reach the concerned scanners during the specific epochs indicated by the crowd flow query. In addition to the influences mentioned for footfall, the closeness of crowd flows to real-life flows of people is influenced by the distance between scanners. Measuring crowd flows between distant scanners must take into account the traveling speed of pedestrians, such that appropriate epochs are chosen. On the other hand, crowd flows between close scanners must take into account the possibility of detecting the same device in multiple places at the same time, because overlapping ranges, though not desirable for crowd monitoring, can happen in practice.

7.2. Dealing with overlapping ranges of scanners

We mentioned in Section 2 that, ideally, scanners are positioned in such a way that their ranges do not overlap. In real-world implementations, though, it can happen that scanners overlap in coverage, either temporarily, due to the fluctuation of ranges, or by design, for contiguous coverage purposes. By following the definition, a crowd flow between such scanners will falsely count devices which find themselves in the overlap as making the transition. The problem is most impactful for crowd flows of complexity $q = 2$, i.e. between two scanners, as having at least a third one would already indicate an actual movement unless all of them overlap, which is uncommon.

In Fig. 19, for scanners with overlapping ranges s_1, s_2 and epochs e_1, e_2, we present the 4 possible situations in which devices might find themselves when counted as part of the crowd flow. All situations show devices detected by s_1 in e_1 and by s_2 in e_2. Situation (a) is similar with the case when scanners have non-overlapping ranges, device a_1 being detected by a single scanner each epoch. In (b) and (c), in one of the epochs devices are detected by both scanners, being in the overlap of ranges. Eventually, devices like a_4 in (d) are detected by both scanners in both epochs.

An improved estimation of the crowd flow in discussion could be achieved, for example, by computing $|\mathcal{D}_{s_1,e_1} \cap \mathcal{D}_{s_2,e_2}| - |\mathcal{D}_{s_1,e_1} \cap \mathcal{D}_{s_2,e_2} \cap \mathcal{D}_{s_1,e_2} \cap \mathcal{D}_{s_2,e_1}|$. This formula excludes from counting devices in (d), for which no information regarding their movement can be derived. Other strategies, such as excluding devices in (b) or (c), can be applied in the same way.

Our system can support such calculations by asking the server to perform the additional necessary BF intersections under encryption, then shuffle and deliver the result to the consumer along with a query response. We tested this on crowd flows between two scanners from the Assen dataset which seemed to have overlapping ranges and indeed we were able to identify and subtract apparently unmoving devices from the statistical counts. To what extent, though, the improved statistical counts are closer to the actual flows cannot be deduced from the available data only. Still, we stress that our system can already support if needed, without modifications, such strategies to deal with overlapping ranges of scanners.

7.3. Comparison with previous work

We have combined BFs with homomorphic encryption for computing statistical counts on crowds on another occasion [13], which we refer to as previous work throughout this section. The current article is a follow-up of that paper, using the same building blocks but proposing a different construction. The main difference lies in where, when and how the query response is built, as well as in how the statistical count is calculated based on that response.

In our previous work the query response is built step by step *on the scanners* concerned by the query, each step a scanner carrying forward only the BF positions indicated by the hashes of identifiers sensed in that epoch, i.e. it multiplies the values found on those positions with an encrypted 1 and it writes an encrypted 0 on all the other positions. Eventually, the final scanner performs set membership testing under encryption, i.e. it multiplies for each sensed identifier the k hash-indicated positions together, assembling the results into a response for the consumer. Then, the consumer learns the statistical count by simply decrypting the response and counting the 1's.

In contrast, in current work the response is built *on the server*, whenever all the necessary data is available, by multiplying position-wise the encrypted BFs and shuffling the result; the statistical count is then estimated by the consumer by applying the proposed equations.

These construction differences have implications on three main dimensions of the system: performance, security and utility. Let us now detail each of them.

Performance. Involving scanners in the creation of responses, as we did in previous work, makes their load increase with the number of queries, eventually leading to a limitation in terms of queries a scanner can be involved into at the same time. Queries in crowd-monitoring systems come at varying pace and they can very well concern the same scanner in an overwhelming way, e.g., when a scanner is at a crossroad, being asked to produce data for numerous crowd flows passing through it. A scanner, which is a hardware device with fixed capabilities, is hard to scale in such settings. Moving most of the computationally expensive operations related to response creation on a server, as we do in current work, ties the load on the scanners to the number of enrolled consumers rather than the number of queries. The number of enrolled consumers does not change often and, when it does, it is predictable. Thus, scanners have stable amounts of computation for long periods of time and it is known beforehand when they must scale. On the other hand, servers are more suitable for dealing with a varying pace of queries and they can scale much easier, as we have shown in 5.2.

Security. Scanners do not know any longer in which queries they are involved. They just perform sensing, write detections in BFs, encrypt and send them to the server, their role in the protocol being reduced. They do not receive any longer other encrypted BFs to perform operations on, which means a lower communication overhead as well as less communication rounds. The server gets more responsibility in the protocol, being trusted to correctly perform operations under encryption and shuffle results before sending them to consumers. Nevertheless, both variants of the protocol are secure under the same honest-but-curious adversary model.

Utility. In previous work, for data minimization purposes, the system was producing data necessary for responses only when it was required by a consumer through a query launched *before* the data collection had to start. While still being able to function this way if desired, the current system offers the additional possibility to launch queries concerning a certain situation also *after* it happened, as there are cases when interest

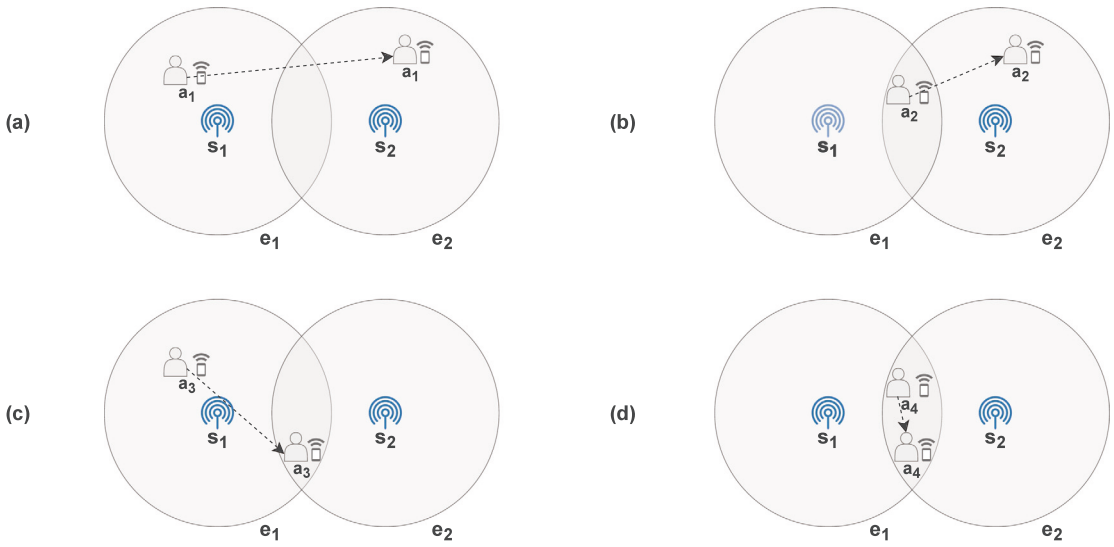


Fig. 19. Situations of crowd flows between scanners with overlapping ranges.

arises post-factum, e.g., for investigating unexpected events. Another point on utility, the accuracy of the statistical counts differs between the two approaches, different ways of calculation being used. In general, it tends to be higher in previous work, as scanners, step by step, inherently drop detections that are not part of the intersection from BFs, this being an effect of their participation in the query response computation when their turn comes, which is no longer the case in current work.

7.4. Security analysis

Our system, as we propose it, is secure against honest-but-curious adversaries, also known as semi-honest. Such adversaries follow the protocol correctly, but they may use the data they handle to learn more about the input of the other parties.

Honest-but-curious scanner. Scanners are assumed tamper-proof in the system model, otherwise there is no way to guarantee the proper functioning of the system. Nevertheless, dropping this assumption for a moment, by compromising a scanner, an honest-but-curious attacker cannot learn more than the input of that scanner in the protocol, that is the detections made by it. It cannot learn detections of other scanners, as no information is exchanged among scanners, neither does it come from the server.

Honest-but-curious server. The server stores and processes EBFs. In order to see what is in there, it would need to be able to decrypt. Decryption is possible by using the private key of a consumer for whom the EBF is produced. Having that private key would mean that the SP colludes with that consumer, which would break the requirements of the system model. Another thing a server may try is encrypting 0's and 1's itself using public keys of consumers and compare them with encrypted values in EBFs, in an attempt to uncover the values stored under encryption. This would be meaningless due to the ciphertext indistinguishability property of the ElGamal scheme, also known as probabilistic encryption, which guarantees that encrypting the same value multiple times results in different ciphertexts.

Honest-but-curious consumer. A consumer can query the system and receive an EBF as answer. After decrypting the answer, knowing that BFs are used in the process, it may be tempted to check whether a certain device is in there by computing the hash functions on its identifier and searching for 1's in the corresponding positions. However, it would be useless since the EBF was shuffled before being sent to the consumer. Thus, the only meaningful information available after decryption is a statistical one, i.e. how many 0's and 1's are in there, which is all that a consumer needs to reach its goal according to the protocol, namely computing statistical counts.

The security of the system can be compromised in case the implementation of the system deviates from the system model. For example, if the non-colluding entities condition is not satisfied, an honest-but-curious server, even without being malicious, would be able to see what is stored in EBFs. Moreover, if the server is malicious, it could very well skip the shuffling part and deliver the EBFs as they are to the consumers.

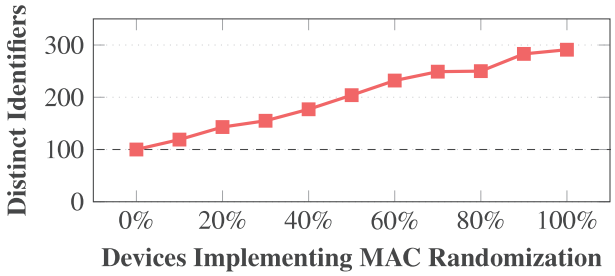


Fig. 20. Simulated example experiment showing the number of distinct identifiers broadcasted by 100 devices in an epoch, considering MAC address randomization. We range the ratio of devices that implement randomization between 0% and 100%, each of them randomly using between 1 and 5 addresses during that epoch. The dashed line indicates the actual number of devices.

7.5. MAC address randomization

Recently, an increasing number of manufacturers of Wi-Fi-enabled devices have started to introduce MAC address randomization. Therefore, when performing probe requests, there are devices that replace their original MAC address with a randomized one, according to each manufacturer's rules.

Our system can produce estimations that accurately resemble the number of devices as long as each input identifier belongs to a single device. Considering that some of the randomization techniques assign more identifiers to a single device, our system may overcount footfall, if, e.g., more random MAC addresses are used by the same device within a single epoch. We illustrate such a situation in Fig. 20. Also, crowd flows may be undercounted if there are devices that re-randomize their addresses between epochs.

To deal with such situations, an additional step could be performed each epoch, as shown in [26] or [27], that would group back the

randomized addresses used by the same device into a single identifier standing for the fingerprint of that device. Each such identifier could be then written in BFs, continuing as presented in the article.

7.6. People carrying multiple devices

Wi-Fi-based crowd-monitoring systems rely on the probe requests transmitted in their range. There could be cases when the number of probe requests may be artificially inflated on purpose, e.g., by a person carrying an abnormally high number of Wi-Fi-enabled devices pretending to be crowds. As we have already mentioned in 7.1, our system counts the distinct identifiers it sees in the probe requests that are collected by Wi-Fi scanners. Therefore, it would not be able, by itself, to discover such cases. We specified in Section 2 that a correction factor (applied afterwards) is expected to deal with differences between the number of sensed devices and the number of people, an aspect which falls outside the scope of this article. However, we are aware that finding a correction factor to also cover the above-mentioned corner cases may not be trivial.

Potential mitigations could include a preprocessing step taking place on scanners, that would take into account, e.g., the physical layer information of signals, and try to detect devices carried by the same person. Then, it would apply some filters on detections or calculate a specific correction factor for those cases. Assuming such mitigations are explored and shown to be feasible, it still remains to be investigated what impact they may have on the accuracy of statistical counts estimated using the preprocessed detections, and what the side effects would be on the privacy protection guarantees offered by the system.

8. Related work

Monitoring crowds of pedestrians has been a matter of study for many years, with several technologies being investigated as promising candidates. The technical capabilities of the Wi-Fi together with its inconspicuous nature of sensing people propelled it as a front runner technology. Therefore, nowadays there are numerous Wi-Fi sensing infrastructures deployed in practice across cities from around the globe. Privacy aspects, however, are not uniformly addressed, often leading to underachievements or hiding pitfalls. In this section we are looking at the state of the art in the area of privacy-preservation in Wi-Fi-based crowd monitoring, exploring the existing limitations.

The process of monitoring crowds using Wi-Fi signals to understand pedestrian dynamics happens in the following way. People traveling in public spaces generally carry with them Wi-Fi-enabled devices, such as smartphones. These devices periodically broadcast Wi-Fi signals in the form of *probe request* frames, searching for available Wi-Fi networks in their vicinity [28]. A sensing infrastructure consisting of Wi-Fi scanners automatically detects devices by continuously capturing the probe requests transmitted within each scanner's range. Based on these detections, interested parties can later on derive relevant information regarding pedestrian dynamics, such as crowd densities and flows [29], as well as mobility patterns occurring within crowds [1].

The key element allowing this to happen is that probe requests are accompanied by the MAC addresses of the devices sending them, serving thus as unique identifiers in the crowd-monitoring process. Problems may arise, though, when it comes to preserving the privacy of individuals, as it was shown that unconsented tracking [30] or even profiling [5] is possible when handling such unique identifiers. In an attempt to prevent such identification from happening, hardware manufacturers introduced MAC address randomization, a mechanism which replaces the real MAC addresses with random ones when sending out probe requests. Despite sometimes being effective, it proved to be insufficient, as works such as [31,32] showed, re-identification remaining possible through several techniques mostly because of unclean and inconsistent deployments across the wide range of manufacturers, which still remains a problem as of 2021 [33].

Wi-Fi crowd-monitoring organizations tried to address the problem as well, mostly by employing pseudonymization, a technique which replaces too the original MAC addresses in probe requests, this time on the capturing side. So-called pseudonyms result after applying either a one-way hash function, a randomized allocation or a deterministic encryption scheme on the original MAC addresses. However, the MAC address space is limited to 2^{48}, so any resulting pseudonyms are susceptible to brute-force attacks, as they are known as weak anonymized data [34]. Furthermore, it was shown by Demir et al. how most commercial solutions using such mechanisms can be broken using off-the-shelf equipment [35], also reconfirmed by Marx et al. [36] in a more recent work.

Finally, there are researchers who looked specifically into protecting the privacy of individuals sensed by Wi-Fi-based crowd-monitoring systems. Kamp et al. [37] proposed a protection mechanism based on linear counting sketches [38]. Their approach does allow estimating footfall in one location as well as reconstructing crowd flows between different locations, privacy protection being based on *expected* k-anonymity over the identifiers, which comes as a natural property of sketches. Also building around k-anonymity, Stanciu et al. [39] introduced detection k-anonymity, an anonymity measure which is, this time, *guaranteed* by an active mechanism for footfall and crowd flow queries while eventually allowing estimations of the concerned crowds. Our approach differs from those previously mentioned in terms of privacy protection, as it loses track of any identifiers once they are encrypted in BFs, later allowing only statistical counts. Another related effort by Allagan et al. [40] makes use of differentially pan-private Bloom filters. While their method works well for large crowds, it achieves low accuracies when handling small crowds, this being a common condition of systems implementing differential privacy.

Other attempts to count crowds using Wi-Fi signals drop the requirement of collecting messages sent by devices [41–43]. Instead, they leverage the impact of walking people on the transmitted radio signals in an area, hence, not requiring the people in the crowd to carry active devices for occupancy estimation. While such methods have no fine-grained knowledge of the people they count (thus offering privacy protection), there is no way to match the people counted in one location with people counted in another location (apart from strictly controlled setups), making their applicability to crowd flows limited.

9. Conclusion

In this work, we propose a novel crowd-monitoring system that produces statistical counts of crowds while fully protecting the privacy-sensitive data of the individuals being monitored. At the core of our solution lies a cryptographic construction in which the detections of individuals are encoded into homomorphically encrypted BFs and then immediately discarded. This construction allows our system to blindly perform computations over encrypted data that it cannot decrypt, such that only statistical counts become available in the clear. Therefore, the system can accommodate, in a privacy-friendly way, footfall counting, as well as counting crowd flows between different locations, measurements otherwise unacceptable due to the risk of uniquely identifying individuals from the handled data.

We implement the system using Raspberry Pi as a scanner and different server configurations as operators under encryption. In addition to simulations, we test the system using real-world data from a large festival. For footfall scenarios concerning the most crowded area of the festival, the accuracy does not get below 97.2% (i.e. 1 device away in that particular case). Also, when measuring crowd flows happening on a circulated street, 88.5% of the statistical counts had an accuracy above 90%, 10.7% between 80% and 90%, 0.6% between 70% and 80% and 3 of them below 70%. For the same crowd flows, 98.7% of the estimations were less than 3 devices away from the real counts. These results demonstrate that highly accurate statistical counts of crowds are indeed practical when dealing with real-world data. Moreover, limited

hardware (i.e. resource-constrained devices as scanners and a laptop as server) proves to be sufficient for this purpose, successfully accommodating a homomorphic encryption scheme on top of probabilistic data structures such as BFs. We hope that our work inspires other researchers searching for a solution in comparable settings, who aim to protect privacy-sensitive data sensed by an infrastructure of data collection points while still being able to use it for the intended purpose of their system.

CRediT authorship contribution statement

Valeriu-Daniel Stanciu: Conceptualization, Methodology, Software, Investigation, Data curation, Writing – original draft, Visualization. **Maarten van Steen:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Ciprian Dobre:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Andreas Peter:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data

References

[1] B. Bonné, A. Barzan, P. Quax, W. Lamotte, WiFipi: Involuntary tracking of visitors at mass events, in: 2013 IEEE 14th International Symposium on "a World of Wireless, Mobile and Multimedia Networks"(WoWMoM), IEEE, 2013, pp. 1–6.

[2] A. Basalamah, Crowd mobility analysis using WiFi sniffers, Int. J. Adv. Comput. Sci. Appl. 7 (12) (2016) 374–378.

[3] M. Dunlap, Z. Li, K. Henrickson, Y. Wang, Estimation of origin and destination information from bluetooth and wi-fi sensing for transit, Transp. Res. Rec. 2595 (1) (2016) 11–17.

[4] H. Hong, C. Luo, M.C. Chan, Socialprobe: Understanding social interaction through passive wifi monitoring, in: Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 2016, pp. 94–103.

[5] M. Cunche, M.-A. Kaafar, R. Boreli, Linking wireless devices using information contained in wi-fi probe requests, Pervasive Mob. Comput. 11 (2014) 56–69.

[6] J. El Mallah, F. Carrino, O. Abou Khaled, E. Mugellini, Crowd monitoring, in: International Conference on Distributed, Ambient, and Pervasive Interactions, Springer, 2015, pp. 496–505.

[7] M. Wirz, T. Franke, D. Roggen, E. Mitleton-Kelly, P. Lukowicz, G. Tröster, Inferring crowd conditions from pedestrians' location traces for real-time crowd monitoring during city-scale mass gatherings, in: 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, 2012, pp. 367–372.

[8] G.D.P. Regulation, Regulation EU 2016/679 of the European parliament and of the council of 27 april 2016, Off. J. Eur. Union (2016).

[9] (2015) https://marketinglaw.osborneclarke.com/advertising-regulation/jc-decaux s-pedestrian-tracking-system-blocked-by-french-data-regulator/. [Online; accessed 05-March-2023].

[10] (2018) https://www.rtvoost.nl/nieuws/303852/Enschede-stopt-tijdelijk-met-wifi-tellingen-na-publicatie-Autoriteit-Persoonsgegevens. [Online; accessed 05-March-2023].

[11] (2018) https://www.emerce.nl/nieuws/wifitellingen-binnensteden-stilgelegd. [Online; accessed 05-March-2023].

[12] (2021) https://autoriteitpersoonsgegevens.nl/nl/nieuws/boete-gemeente-enschede-om-wifitracking. [Online; accessed 05-March-2023].

[13] V.-D. Stanciu, M.v. Steen, C. Dobre, A. Peter, Privacy-preserving crowd-monitoring using bloom filters and homomorphic encryption, in: Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, 2021, pp. 37–42.

[14] B. Soundararaj, J. Cheshire, P. Longley, Estimating real-time high-street footfall from wi-fi probe requests, Int. J. Geogr. Inf. Sci. 34 (2) (2020) 325–343.

[15] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.

[16] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, Y. Tang, On the false-positive rate of bloom filters, Inform. Process. Lett. 108 (4) (2008) 210–213.

[17] S.J. Swamidass, P. Baldi, Mathematical correction for fingerprint similarity measures to improve chemical retrieval, J. Chem. Inf. Model. 47 (3) (2007) 952–964.

[18] G. Bianchi, L. Bracciale, P. Loreti, "Better than nothing" privacy with bloom filters: To what extent? in: International Conference on Privacy in Statistical Databases, Springer, 2012, pp. 348–363.

[19] R.L. Rivest, L. Adleman, M.L. Dertouzos, et al., On data banks and privacy homomorphisms, Found. Secur. Comput. 4 (11) (1978) 169–180.

[20] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inf. Theory 31 (4) (1985) 469–472.

[21] A. Appleby, MurmurHash3, 2016, URL https://github.com/aappleby/smhasher/wiki/MurmurHash3.

[22] O. Papapetrou, W. Siberski, W. Nejdl, Cardinality estimation and dynamic length adaptation for bloom filters, Distrib. Parallel Databases 28 (2) (2010) 119–156.

[23] M. Adalier, A. Teknik, Efficient and secure elliptic curve cryptography implementation of curve P-256, in: Workshop on Elliptic Curve Cryptography Standards, Vol. 66, 2015.

[24] Y. Ejgenberg, M. Farbstein, M. Levy, Y. Lindell, SCAPI: The secure computation application programming interface, IACR Cryptol. ePrint Arch. 2012 (2012) 629.

[25] A.-C. Petre, C. Chilipirea, M. Baratchi, C. Dobre, M. van Steen, WiFi tracking of pedestrian behavior, in: Smart Sensors Networks, Elsevier, 2017, pp. 309–337.

[26] M. Uras, E. Ferrara, R. Cossu, A. Liotta, L. Atzori, MAC address de-randomization for WiFi device counting: Combining temporal-and content-based fingerprints, Comput. Netw. 218 (2022) 109393.

[27] P. Torkamandi, L. Kärkkäinen, J. Ott, An online method for estimating the wireless device count via privacy-preserving wi-fi fingerprinting, in: Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29–April 1, 2021, Proceedings 22, Springer, 2021, pp. 406–423.

[28] A. Musa, J. Eriksson, Tracking unmodified smartphones using wi-fi monitors, in: Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems, 2012, pp. 281–294.

[29] L. Schauer, M. Werner, P. Marcus, Estimating crowd densities and pedestrian flows using wi-fi and bluetooth, in: Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 2014, pp. 171–177.

[30] M. Cunche, I know your MAC address: Targeted tracking of individual using wi-fi, J. Comput. Virol. Hacking Tech. 10 (4) (2014) 219–227.

[31] M. Vanhoef, C. Matte, M. Cunche, L.S. Cardoso, F. Piessens, Why MAC address randomization is not enough: An analysis of wi-fi network discovery mechanisms, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, 2016, pp. 413–424.

[32] J. Martin, T. Mayberry, C. Donahue, L. Foppe, L. Brown, C. Riggins, E.C. Rye, D. Brown, A study of MAC address randomization in mobile devices and when it fails, Proc. Priv. Enhanc. Technol. 2017 (4) (2017) 365–383.

[33] E. Fenske, D. Brown, J. Martin, T. Mayberry, P. Ryan, E. Rye, Three years later: A study of MAC address randomization in mobile devices and when it succeeds, Proc. Priv. Enhanc. Technol. 2021 (3) (2021) 164–181.

[34] L. Demir, A. Kumar, M. Cunche, C. Lauradoux, The pitfalls of hashing for privacy, IEEE Commun. Surv. Tutor. 20 (1) (2017) 551–565.

[35] L. Demir, M. Cunche, C. Lauradoux, Analysing the privacy policies of wi-fi trackers, in: Proceedings of the 2014 Workshop on Physical Analytics, 2014, pp. 39–44.

[36] M. Marx, E. Zimmer, T. Mueller, M. Blochberger, H. Federrath, Hashing of personally identifiable information is not sufficient, in: SICHERHEIT 2018, Gesellschaft für Informatik eV, 2018.

[37] M. Kamp, C. Kopp, M. Mock, M. Boley, M. May, Privacy-preserving mobility monitoring using sketches of stationary sensor readings, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 370–386.

[38] F. Rusu, A. Dobra, Statistical analysis of sketch estimators, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, 2007, pp. 187–198.

[39] V.-D. Stanciu, M. van Steen, C. Dobre, A. Peter, K-anonymous crowd flow analytics, in: MobiQuitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 2020, pp. 376–385.

[40] M. Alaggan, M. Cunche, S. Gambs, Privacy-preserving wi-fi analytics, Proc. Priv. Enhanc. Technol. 2018 (2) (2018) 4–26.

[41] S. Depatla, Y. Mostofi, Occupancy analytics in retail stores using wireless signals, in: 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), IEEE, 2019, pp. 1–9.

[42] S. Depatla, Y. Mostofi, Passive crowd speed estimation in adjacent regions with minimal WiFi sensing, IEEE Trans. Mob. Comput. 19 (10) (2019) 2429–2444.

[43] M. Bocca, O. Kaltiokallio, N. Patwari, S. Venkatasubramanian, Multiple target tracking with RF sensor networks, IEEE Trans. Mob. Comput. 13 (8) (2013) 1787–1800.