Computer Communications 125 (2018) 13–23

Contents lists available at ScienceDirect

Computer Communications

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

Efficient privacy-preserving implicit authentication

Alberto Blanco-Justicia, Josep Domingo-Ferrer*

Universitat Rovira i Virgili, Department of Computer Science and Mathematics, CYBERCAT-Center for Cybersecurity Research of Catalonia, UNESCO Chair in Data Privacy, Av Països Catalans 26, Tarragona, Catalonia E-43007

ARTICLE INFO

Keywords:
Privacy-preserving implicit authentication
Bloom filters
Privacy

ABSTRACT

The number of online service accounts per person has rapidly increased over the last years. Currently, people have tens to hundreds of online accounts on average, and it is clear that users do not choose new, different, and strong passwords for each of these accounts. On the other hand, it is quite inconvenient for the user to be forced to explicitly authenticate each time she wants to use one of her many accounts; this is especially true with small user devices like smartphones. Implicit authentication is a way to mitigate the preceding problems by authenticating individuals based not only on their identity and credentials, but on how they interact with a device, *i.e.* their behavior. User behavior can be characterized by collecting keystroke patterns, browser history and configuration, IP addresses and location, among other characteristics of the user. However, keeping the user's behavior profile in authentication servers can be viewed as privacy-invasive. Privacy-preserving implicit authentication has been recently introduced to protect the privacy of the users' profiles, specifically against the party performing the authentication, which we call the server in the sequel. Yet, the privacy-preserving implicit authentication schemes proposed so far involve substantial computation both by the user and the server. We propose here a practical mechanism based on comparing behavior feature sets encoded as Bloom filters. The new mechanism entails much less computation and can accommodate much more comprehensive sets of features than previous alternatives.

1. Introduction

Implicit authentication refers to a software system authenticating individuals based on the way they interact with their device, *i.e.* their behavior. In this context, the user behavior can be determined by collecting a variety of features, such as keystroke patterns, browser history and configuration, IP addresses, location, visible antennas, etc. Implicit authentication is a *complement*, rather than a substitute, of the usual explicit authentication based on identifiers and/or credentials. Authenticating implicitly can make life easier for users by reducing the number of times they have to authenticate explicitly.

Implicit authentication is gaining importance as the smartphone market rises. Relatively small and sometimes unwieldy screen keyboards in smartphones make typing strong passwords a difficult task. This situation, added to the well-known problem of weak password choices, repeatedly aired in the media, makes the use of secondary authentication mechanisms almost mandatory. Among these, biometric (fingerprint) authentication and two-factor authentication with one-time passwords are the most common choices. Biometric authentication has the shortcomings of needing special sensors in the user's device and requiring the authenticating server to acquire and store the user's reference biometric pattern. Two-factor authentication, on its side, has an intrinsic problem: the second channel (email, SMS, mobile app) used for confirmation is usually accessible on the same device (typically an Internet-enabled smartphone) used for the primary channel, so both channels may be simultaneously compromised.

Implicit authentication is not free of problems either. A salient issue is the privacy exposure of end users, who need to be profiled in order to provide a reference pattern against which their current behavior can be authenticated by the server. To keep the user's profile private against the server, researchers have proposed privacy-preserving implicit authentication [6,18]. In these proposals, the user's reference profile is stored in encrypted form at the server and the fresh usage sample captured by the user's device is compared against that reference profile. While this solves the privacy issues, it entails substantial computation, both by the user's device and the server.

1.1. Contribution and plan of this paper

We propose a computationally efficient privacy-preserving implicit authentication mechanism in which only fingerprints of the users' usage profiles are revealed. We use Bloom filters to encode the user's

* Corresponding author.
E-mail addresses: alberto.blanco@urv.cat (A. Blanco-Justicia), josep.domingo@urv.cat (J. Domingo-Ferrer).

https://doi.org/10.1016/j.comcom.2018.04.011
Received 16 November 2017; Received in revised form 9 March 2018; Accepted 13 April 2018
Available online 27 April 2018

reference profile and we leverage the properties of Bloom filters to compute the distance between the stored reference profile and the fresh samples provided by the user. The privacy of users' profiles is protected as long as cryptographic hash functions are secure.

Our proposed mechanism produces fingerprints of the feature sets that are compact and can be easily integrated in existing authentication protocols, for example, as headers in HTTP packets.

Section 2 gives an overview of related subjects, including implicit authentication systems and privacy-preserving implicit authentication systems. Section 3 describes the adversarial model. Section 4 describes Bloom filters in detail. Section 5 recalls the types of features included in users' profiles, and how to compute the dissimilarity between sets of features, depending on their type. Section 6 presents our proposed mechanism. Section 7 discusses the privacy and the security of our proposal. Finally, Section 8 analyzes the accuracy and the performance of the new system. Conclusions and future research lines are gathered in Section 9.

2. Related work

2.1. Implicit authentication

In implicit authentication, a server can authenticate users by checking whether their behavior is compatible or similar enough to their past recorded behavior. In this context, the user's behavior can be modeled as a combination of features like her browsing history, usual location, keystroke patterns, usually visible cell stations, etc. A user profile consists of one or several such sets of features.

In [11], empirical evidence was given that the features collected from the user's device history are effective to distinguish users and therefore can be used to implicitly authenticate them (instead of or in addition to explicit authentication based on the user's providing a password). Muncaster and Turk propose a general framework for continuous authentication of users in [16], based on the integration of active (*e.g.* fingerprints) and passive (*e.g.* keystroke patterns) biometric measurements. Dynamic Bayesian networks are used to aggregate the classification decisions and scores from the different biometric authentication mechanisms. They demonstrate the framework with face recognition and keystroke pattern analysis. Clarke and Furnell explore authentication using keystroke patterns analysis in [5]. The authors use neural networks to decide whether the users interacting with the smartphones' keyboards are the rightful owners of the devices. The authors acknowledge, however, that keystroke analysis is too dependent on the specific user (not all users use the keyboard enough, and some vary their writing patterns constantly), and so multimodal approaches seem a better solution. The authors in [19] propose SenGuard, an implicit authentication mechanism for mobile devices, which uses information from the touch screen, location, means of transport and voice patterns. Authentication decisions are taken with a space-time multi-modality classifier.

These proposals are designed as local authentication mechanisms to the mobile device and do not need to take privacy into account, since data are not meant to leave the local device. However, we argue that storing the profile in the user's device is insecure, because an intruder may gain access to it and learn sensitive information about the user, or even impersonate her. Therefore, it is safer to store the users' profiles in a secure facility, for example in the provider's premises. However, a user profile includes potentially sensitive data, and storing it outside the user's device violates her privacy.

This privacy risk is only partly mitigated by using a third party to store the users' profiles, for example the ISP or carrier. The typical architecture in this case consists of the user's device, a service provider and the carrier.

2.2. Privacy-preserving implicit authentication

In the privacy-preserving implicit authentication system proposed by Safa et al. [18], the user's device encrypts the user's usage profile at set-up time, and forwards it to the carrier, who stores it for later comparison. In this case, the security problem outlined in the previous section is fixed because the profile is never stored in the user's device (it is collected, encrypted, sent and immediately deleted by the device). Likewise, the privacy problem is also solved, because the profile sent to the carrier is encrypted. In fact, since the user's profile is exported in encrypted form, strictly speaking the carrier is no longer needed as a third party to store profiles and conduct the authentication: both functions could be performed by the service provider himself. Therefore, we will name the authenticating party as the server, which can be the carrier or the service provider.

The core of [18] is the algorithm for computing the dissimilarity score between two inputs: the fresh sample provided by the user's device and the profile stored at the server. All the computation takes place at the server and both inputs are encrypted: indeed, the server stores the encrypted profile and the user's device sends the *encrypted* fresh sample to the server. Note that the keys to both encryptions are only known to the user's device (it is the device that encrypts everything).

The server computes a dissimilarity score at the feature level, while provably guaranteeing that: (i) no information about the profile stored at the server is revealed to the device other than the average absolute deviation of the stored feature values; (ii) no information about the fresh feature value provided by the device is revealed to the server other than how it is ordered with respect to the stored profile feature values.

The score computation protocol uses two different encryption schemes: a homomorphic encryption scheme *HE* (for example, Paillier) and an order-preserving symmetric encryption scheme *OPSE*. This protocol is restricted to numerical features, due to the kind of computations that need to be performed on them. Such a limitation is a shortcoming, because behavior characterization may require non-numerical features, *e.g.* the browser history.

The implicit authentication system proposed by Domingo-Ferrer et al. [6] tries to overcome some of the limitations of the previous approach: it uses a single cryptosystem, it does not leak the order of fresh sample values, it does not leak the average absolute deviation of the stored feature values, and it can deal with non-numerical features. To do so, it builds on the work done by Blanco-Justicia et al. [2], which proposes a mechanism to compute the distance between preference functions, defined as sets of independent categorical features, correlated categorical features, or independent numerical features.

In [6], the user's device encodes and encrypts the user's profile using the Paillier cryptosystem and sends it to the server, along with some auxiliary values. Neither the encrypted set nor the auxiliary values reveal anything about the user's profile, except for the size of the set. Moreover, the user only keeps a secret auxiliary value and deletes the Paillier secret key, so the profiles cannot be recovered from the device either. To authenticate, the user's device sends an encrypted fresh sample of her activity to the server and then user and server engage in a two-party protocol to compute the distance between the stored or reference profile and the new sample. Note that the results are never decrypted, but checked using the server's and the user's auxiliary values.

Although Domingo-Ferrer et al. [6] solves several shortcomings of [18], it remains very demanding in computational terms. In fact, computing the auxiliary values during set-up requires inverting matrices, which in practice limits the maximum allowable size of the feature sets (due to computing time constraints).

In [22], the authors use a well-known approach in data mining, *i.e.* dimensionality reduction, to reduce the information revealed by the profiles, while keeping enough information for profiles to be correctly classified. This proposal, though, is limited to numerical features and

does not entirely hide profile information.

In this new proposal, we capitalize on the work done in [2,6]: while we continue to compute the distances between the profiles via set intersection, we take the novel approach of doing so using Bloom filters to relieve the computational complexity.

3. Problem statement

3.1. Scenario

We consider a scenario in which smartphone users log into online services offered by some service provider. Users set their login information at registration time, and this information is managed by either the service provider or by a third party (an identity provider that delivers authentication services). The entity in charge of authenticating users offers an additional security measure: it analyzes the behavior of users to detect potentially compromised user accounts. The service provider denies service to accounts labeled as compromised, and the rightful owners of those accounts are notified.

As mentioned in the introduction above, implicit authentication complements explicit authentication (based *e.g.* on username-password) by trying to minimize the number of times users are asked to input their credentials. During an initial or training phase after the user registers to the service, the smartphone starts sending protected user profiles to the service provider. While in this training phase, the service provider will keep requesting the user credentials as usual. This phase is also used to set the threshold to accept an authentication attempt by the user. When the training phase ends, the service provider will stop requesting the user credentials unless implicit authentication fails, in which case it will request them. This procedure is similar to those already implemented by Google, Steam and others: when the user logs in from a new device (*i.e.* a device from which the user has never logged in to the service before), the service provider requests the user credentials.

3.2. User profiles, behavior samples and feature sets

In this section, we describe the user profiles as we will use them throughout the rest of this work. A **user profile** is a list of snapshots of the user's behavior at different times. These snapshots, or **samples**, are labeled by a timestamp and contain one or more feature sets. Each **feature set** contains readings from one specific data source in the user's device for a fixed period of time.

Fig. 1 shows an example of a user profile $\mathbb{P}$ with two samples $\mathcal{S}_1$ and $\mathcal{S}_2$, each of them containing several feature sets S_i (each of them labeled according to the data it contains), namely installed applications, visible cell towers, web browsing history and visited locations. The label $\mathcal{S}_t$ refers to specific sample and the subscript t may refer to the time of collection (a timestamp). Each of the feature sets includes information gathered from different sources during some predefined period of time; for example, the set labeled as CT in sample $\mathcal{S}_1$ may be the set of cell towers that the device has seen during the last day.

Typical data sources considered for authentication are, among others, installed applications, installed applications by category, usage of applications, usage of applications by category, visible cell towers, strength of the signal of cell towers, battery level at the time the device is connected for charging, time between consecutive charges, power consumption, idle and awake times, web browsing history, web browsing history by category, and trajectories of the user. Some of these features may be considered more important than others when taking an authentication decision; hence, different weights may be assigned to the various features.

In the implicit authentication protocol, the user sends a new sample to the server, who compares it to ℓ previous samples, feature set by feature set, to reach an authentication decision. It is also worth considering that, since users may behave differently in work days and weekends, samples may need to be compared with parts of the profile

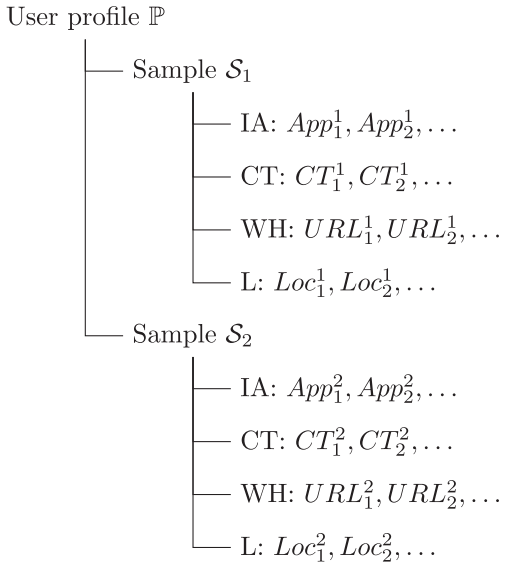

Fig. 1. User profile with two behavior samples. Within each sample, feature sets are as follows: installed applications (IA), visible cell towers (CT), web browsing history (WH), visited locations (L).

corresponding to similar days and times. New samples that result in successful authentications can be stored by the server to update the user profile.

Assuming that every feature has the same weight when taking the authentication decision (*e.g.*, installed applications matter as much as visited locations), our construction allows representing each sample as a single feature set of the following form (where each value is labeled with the name of the feature set it comes from):

- $\mathcal{S}_1$: {IA:App_1^1, IA:App_2^1, $\cdots$, CT:CT_1^1, $\cdots$, WH:URL_1^1, $\cdots$, L:Loc_1^1, $\cdots$}
- $\mathcal{S}_2$: {IA:App_1^2, IA:App_2^2, $\cdots$, CT:CT_1^2, $\cdots$, WH:URL_1^2, $\cdots$, L:Loc_1^2, $\cdots$}

In this case, the implicit authentication protocol can run a single set comparison per sample. However, if the weights of the features differ, or the profiles contain categorical *and* numerical feature sets, the authentication protocol will have to compute the distances between several sets. An example of this situation is shown in Fig. 2.

In this case, to compare the sample S_3 with older samples (of the same form) in an authentication attempt, the following sets have to be compared:

- {IA:App_1^3, IA:App_2^3, $\cdots$, CT:CT_1^3, CT:CT_2^3, $\cdots$}
- {WH:URL_1^3, WH:URL_2^3, $\cdots$, L:Loc_1^3, L:Loc_2^3, $\cdots$}
- {K_1^3, K_2^3, $\cdots$}

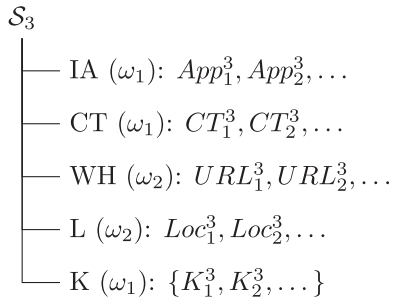

Fig. 2. Behavior sample with different weights (denoted by ω_1 and ω_2). Categorical feature sets: installed applications (IA), visible cell towers (CT), web browsing history (WH), visited locations (L). Numerical feature set: kilometers walked each hour (K).

While the mechanisms for the collection of data and the data sources themselves are outside of the scope of this work, we impose the requirement that features must be discretized, for example via generalization/coarsening. Take location data as an example: GPS location data are too fine-grained for comparison using our distance computation mechanism, so we require them to be coarsened either by translating the latitude-longitude pairs to names (*e.g.* street names), or by truncating some decimals (which in practice amounts to defining a grid in a map).

3.3. Privacy attacker

We regard any entity with legitimate or illegitimate access to the stored user profiles as a potential privacy attacker. An attacker seeing the profile of a user learns sensitive information about the user, such as her typical locations, her preferences, the software installed on her computer, etc. The user's profile may also allow inferring the user's identity, in case the latter is not readily available. Hence, user profiles must be protected while in transit and when stored on the server premises.

3.4. Impersonator

An attacker who gains access to a user's device, her account credentials (*e.g.* login and password) or both, may try to impersonate the user by accessing her accounts. By analyzing the behavior of the impersonator it should be possible to detect the attack and deny access to the impersonator.

4. Bloom filters

Bloom filters [3] are probabilistic space-efficient data structures that encode datasets while still allowing membership queries. But the appeal of Bloom filters goes beyond efficient set encoding and membership checking: they also allow computing set unions and intersections, and therefore the Jaccard distance between sets, while offering security guarantees based on those of cryptographic hash functions.

A Bloom filter consists of a bit array $B = b_0, \cdots, b_{m-1}$ of length m, with all bits initially set to 0, and is equipped with k different hash functions with range $[0, ..., m-1]$ each of which maps some set element to one of the m array positions with a uniform random distribution. Typically $k \ll m$. An element e to be inserted is hashed with the k hash functions, and the corresponding bits $b_{h_0(e)}, \cdots, b_{h_{k-1}(e)}$ are set to 1. Accordingly, a membership query for element e is performed by checking whether $b_{h_0(e)} = \cdots = b_{h_{k-1}(e)} = 1$.

Membership queries to a Bloom filter can certainly return false positives: the query answer can say that an element not in the set belongs to it. However, proper parameter setting (in particular increasing m), can make the false positive rate arbitrarily low, as it follows from the expression of the probability of false positive: if a Bloom filter contains n elements, this probability is given by

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k. \tag{1}$$

The above expression can be viewed as the probability that the indices obtained by hashing the element with the k hash functions are all set to 1, either by another single element (which is unlikely, because it would mean that the k hash values of the second element collide with those of the first one) or by a combination of other inserted elements. Note that Expression (1) is only an approximation, because it assumes that a bit is set to 1 independently of the values of the other bits, which is a simplification of reality [4].

The number of elements encoded in a Bloom filter (*i.e.* the cardinality of the encoded set S) is approximated as

$$|S| \approx -\frac{m}{k} \ln\left(1 - \frac{H(B_S)}{m}\right), \tag{2}$$

where B_S is the bit array resulting from encoding S, $H(\cdot)$ is the Hamming weight of a bit array (number of bits set to 1), m is the length of the Bloom filter and k is the number of hashes [21].

4.1. Bloom filters in privacy-preserving data mining

Private matching schemes are an interesting multi-party computation primitive in privacy-preserving data mining. They are useful for relational equi-joins and intersections of databases privately held by different entities that do not want to disclose their respective databases to each other. They can also play a role when searching some text in documents, matching preferences in social networks, or comparing user profiles. In this paper, we use them for the latter purpose.

Among the various implementations of private matching schemes in the literature, the work in [8] is perhaps one of the most relevant. In this work, the authors propose several protocols for private computation of set intersections and set intersection cardinalities under different adversarial models. In these protocols, the sets are encoded as polynomials and encrypted under the Paillier cryptosystem, whose homomorphic properties allow for the evaluation of encrypted polynomials. In [2], the authors build upon these protocols to compute distances between private functions, including numerical ones. These protocols, however, rely on expensive cryptographic operations and thus are not practical for very big datasets.

In [7], the authors propose a private matching scheme based on garbled Bloom filters and oblivious transfer that overcomes the scalability problem of previous proposals. In this protocol, the datasets are encoded as garbled Bloom filters (an extension of Bloom filters). The parties then engage in an oblivious transfer protocol to compute a new Bloom filter that encodes the intersection of the original sets. Pinkas et al. [17] improve on the efficiency of the previous protocol. The mechanism described in [14] uses the homomorphic properties of the Goldwasser–Micali cryptosystem to test membership of elements in encrypted Bloom filters (the Bloom filter is encrypted bit by bit). Schnell et al. [20] also use Bloom filters to link records from vertically partitioned data with encrypted identifiers.

Bloom filters have been used in other applications of privacy-preserving data mining; for example, to implement a secure dot product protocol to derive association rules from vertically partitioned data [12].

4.2. Union and intersection of sets

The union and intersection of encoded sets can be easily computed by performing the bitwise $\vee$ (or) and $\wedge$ (and) operations, respectively. If we assume the sets A and B and their respective encodings B_A and B_B are of the same size m, with B_A and B_B having both k hash functions, the Bloom filter

$$B_{A \cap B} = b_{A,0} \wedge b_{B,0}, \ ..., b_{A,m-1} \wedge b_{B,m-1}$$

represents the encoding of the intersection set $A \cap B$. Likewise, the union set can be obtained by computing the bitwise $\vee$ operation.

Note that a Bloom filter obtained by encoding $A \cap B$ or $A \cup B$ will not be exactly equal to the Bloom filters obtained by performing the above operations. This is due to the underlying probabilistic nature of Bloom filters, and may cause some elements to be lost or the false positive rate to increase.

Finally, by applying Expression (2) to the resulting Bloom filters, we can obtain estimates of the cardinalities of the union and the intersection of sets. We plan to use the cardinality of the intersection of two sets to compute their distance.

4.3. Security and privacy of Bloom filters

Gerbet et al. [9] give security definitions for Bloom filters, by deriving them from the standard security properties of cryptographic hash functions.

Definition 1 (*Support*). The support of a vector B of size m, denoted as $\mathsf{supp}(B)$, is the set of its non-zero coordinate indices:

$$\mathsf{supp}(B) = \{i \in [0, \dots, m-1],\ b_i \neq 0\}.$$

Definition 2 (*Pre-image of a Bloom filter*). Given a Bloom filter B, a pre-image of the filter is a string $y \in \{0,\ 1\}^*$ with indices $I_y = \{h_0(y), \dots, h_{k-1}(y)\} \subseteq \mathsf{supp}(B_x)$.

In a *pre-image attack*, an adversary is given B. Finding pre-images becomes easier as the support of the filter increases (as a consequence of the insertion of more elements). In the extreme case in which the size of the support equals the length m of the filter, finding a pre-image is trivial. This is why *we define a pre-image attack to be successful only if the attacker finds a pre-image y that coincides with an element inserted in B*. Even if the attacker does not know the elements inserted in B, she may be able to recognize one when she hits it, due to any redundancy in the element (for example, an URL is easy to recognize).

Definition 3 (*Second pre-image of a Bloom filter*). Given a Bloom filter B_x containing an element $x \in \{0,\ 1\}^*$ with indices $I_x = \{h_0(x), \dots, h_{k-1}(x)\}$, a second pre-image of the filter is another string $y \neq x$ with $I_y = \{h_0(y), \dots, h_{k-1}(y)\}$ such that $I_y \subseteq \mathsf{supp}(B)$.

In a *second pre-image attack*, an adversary is given B_x and its element x. The attack succeeds if the attacker finds a second pre-image $y \neq x$ that also passes the membership test.

Definition 4 (*Collision in a Bloom filter*). Given a Bloom filter B, two strings $x \in \{0,\ 1\}^*$ and $y \in \{0,\ 1\}^*$, with $x \neq y$, $I_x = \{h_0(x), \dots, h_{k-1}(x)\}$, and $I_y = \{h_0(y), \dots, h_{k-1}(y)\}$ are a collision if both $I_x \subseteq \mathsf{supp}(B)$ and $I_y \subseteq \mathsf{supp}(B)$.

In a *collision attack*, an adversary is given B. The attack succeeds if the adversary finds a collision.

In the system we propose, servers store the users' profiles encoded as Bloom filters. Therefore:

- In order to preserve the privacy of the users, it is important that the server, or any attacker with access to the server, cannot obtain the profiles of the users from their encoded versions. Thus, we need Bloom filters to resist pre-image attacks in order to ensure the privacy of the users.
- On the other hand, resistance to second pre-image attacks is needed to ensure that the server or any attacker cannot replace a known legitimate profile by a fabricated profile that also passes the authentication test.
- Finally, resistance to collision attacks is needed to ensure that it is not easy for the attacker to find a random profile that passes the authentication test as if it was some legitimate profile (unknown to the attacker).

Gerbet et al. [9] also provide the computational complexities of pre-image and second pre-image attacks against Bloom filters, and point out that typical implementations of Bloom filters use non-cryptographic hash functions for efficiency reasons and are thus weak against these attacks. They recommend using keyed hash functions, such as message authentication codes (*e.g.* HMAC) to thwart attacks against Bloom filters. Using HMAC does not only increase the domain of the hash function (the HMAC input is augmented with the key), but also makes the computation slower, which is a desirable property to keep hash functions secure against brute-force search.

4.4. Secure instantiation of the Bloom filters

We follow the construction in [15] to build our Bloom filters. Basically, the k hash functions $h_i(x)$ are constructed from two independent hash functions $g_1(x)$ and $g_2(x)$ as $h_i(x) = g_1(x) + ig_2(x) \bmod m$. This strategy reduces the computing time to encode the sets, while maintaining the false positive rate as low as if k independent hash functions were used. In addition, to ensure that the domain of the $h_i(x)$ is large, we take $g_2(x)$ to be keyed, that is, $g_2(x, \mathsf{key})$, so that $h_i(x)$ is also keyed, that is, $h_i(x, \mathsf{key})$.

The optimal values for m and k can be computed as a function of the maximum number N of allowed element insertions, and the maximum acceptable false positive rate ρ [9]:

$$m = -\frac{N \ln \rho}{(\ln 2)^2}, \qquad k = \frac{m}{N} \ln 2. \tag{3}$$

5. Dissimilarity of sets

In this section, we describe methods to compute the dissimilarity between sets from the size of their intersection. We draw on the work in [2], and so we remain close to their terminology. In that article, three types of feature sets are considered: independent categorical feature sets, correlated categorical feature sets, and independent numerical feature sets. We will limit our description and protocol to the first and third cases, because the second case requires supporting computation of intersections between multisets, something that we cannot accomplish with standard Bloom filters.

5.1. Independent categorical feature values

Consider two sets X and Y containing independent categorical values, such that the relationship between any two values is equality or nothing. These sets might for example represent the user's browser history (containing only domain names, but not specific pages, because specific pages would be clearly correlated to their domains), visible cell towers, or installed applications. The (dis)similarity between X and Y can be computed as the multiplicative inverse of the size of their intersection, that is $1/|X \cap Y|$, or ∞ when the intersection is empty. Note that Bloom filters allow computing also the cardinality of the union of two sets, which makes it possible, in our case, to compute the Jaccard similarity index $J(X, Y) = |X \cap Y|/|X \cup Y|$ or its complement, the Jaccard distance, that is $d_J(X, Y) = 1 - J(X, Y) = (|X \cup Y| - |X \cap Y|)/|X \cup Y|$. This latter distance, being normalized, is a very convenient measure.

Clearly, the more the coincidences between X and Y, the more similar is the profile stored at the server to the fresh sample collected by the device.

5.2. Independent numerical feature values

In this case, the profile of the user is a set of numerical values, for example, sensor data, the browser history expressed as the number of accesses to each website in a list, a histogram of user preferences, etc.

Given two sets $X = \{x_1, \cdots, x_t\}$ and $Y = \{y_1, \cdots, y_t\}$, a way to measure the dissimilarity between them is to compute $\sum_{i=1}^{t} |x_i - y_i|$. If X and Y represent normalized histograms, that is, $0 \leq x_i \leq 1$ for all i and $\sum_{i=1}^{t} x_i = 1$ (or 100 if the values are given as percentages), we could also normalize the resulting distance, because the maximum possible distance is known (2 if the features of each histogram add to 1).

6. Privacy-preserving implicit authentication using Bloom filters

In this section we describe the system we propose to implicitly authenticate users from their recorded behavior. We use Bloom filters to encode the users' profiles so that the server cannot obtain the profile of

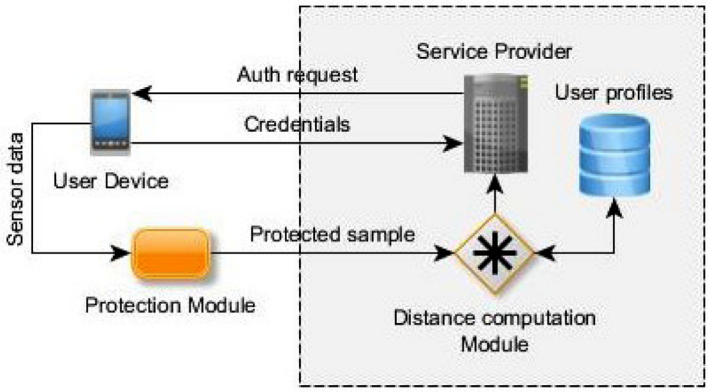

Fig. 3. Basic architecture.

a user from its Bloom filter representation; this guarantees privacy. The decision whether a user is authenticated or not will be taken by computing the distance between previously recorded profiles and the fresh samples the user provides to the server; authentication will be positive if that distance is below a predefined threshold.

First, we give a high-level vision of the architecture. Then, we specify the authentication protocol.

6.1. Architecture

The high-level architecture of our implicit authentication mechanism is centered on: (i) a protection module in the user's smartphone that collects sensor data and protects the samples; and (ii) a distance computation module on the server's side that can compute the (dis)similarity of the encoded fresh sample and the encoded recorded user profile, and return an authentication score or decision. We propose two alternative architectures built on these core modules. The first one, shown in Fig. 3, is a traditional approach, in which the service provider is in charge of authenticating its users; the second one, shown in Fig. 4, follows a single sign-on approach, in which an identity provider, for example the carrier, is in charge of authenticating the users on behalf of the service provider.

Next, we briefly describe the two above-mentioned core modules. Then, the interaction between the protection module and the distance computation module, that is, the implicit authentication protocol, is described in Section 6.2.

We assume in the sequel that the user's device is not infected by a malware capable of reading sensor data. This is a reasonable assumption, because an infected user's device could leak the user's behavior anyway, so there would be no point in caring to use privacy-preserving implicit authentication.

If one wishes protection against malware at the user's device, then the device's sensor data should be only available in privileged execution mode (*e.g.* ARM's TrustZone [1]) and our authentication mechanism should run also in this mode.

6.1.1. Protection module

The protection module in the user's device is a software module that first gathers data from the device. These data are used to build the fresh sample of the user profile which will be the input to implicit authentication. Then the protection module protects the sampled profile by encoding it into a Bloom filter. Finally, the module sends the Bloom filter to the server.

6.1.2. Distance computation module

The distance computation module runs on the server side and is capable of comparing protected behavior samples against previously stored protected reference user profiles. The comparison returns an authentication score that is compared with a threshold, in order to output an authentication decision.

6.2. Implicit authentication protocol

6.2.1. Set-up protocol

Let the initial user's profile be $\mathbb{P} = \emptyset$. The aim of the set-up protocol is to begin populating this profile with one or several behavior samples. Let the first sample $\mathscr{S}_0$ be a collection of sets $\mathscr{S}_0 = \{S_{0,i} \mid 1 \leq i \leq p\}$, in which every set $S_{0,i} = \{s_1^{0,i}, \ldots, s_n^{0,i}\}$ is a labeled feature set, for $n \leq N$. The maximum allowed value N, as well as the values k and m (number of hash functions and length of the Bloom filters, respectively), are set by the server (who may be the carrier or the service provider). The weights of each of the feature sets are also set by the server.

The sample $\mathscr{S}_0$ and every subsequent sample $\mathscr{S}_t$ to be added to the user's profile are preprocessed as follows:

- Sets of categorical features $S_{t,i} \in \mathscr{S}_t$ are aggregated as described in Section 3.2, by concatenating the label of the feature set to each of the feature values and uniting them into sets $R_{t,\ j}$, taking into

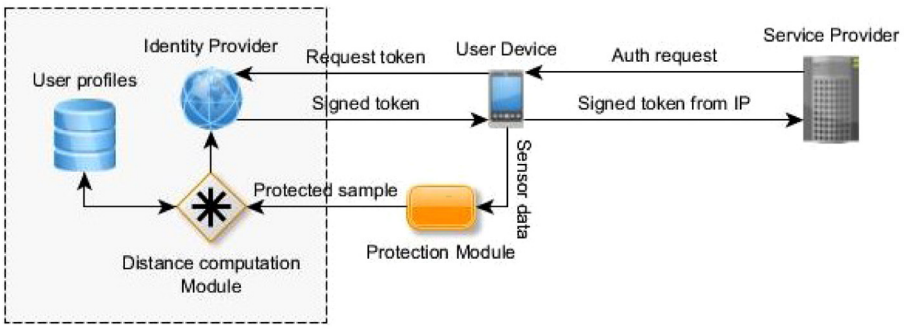

Fig. 4. Architecture with an identity provider (IP).

account the weights of each of the features. For example, sets *Applications*: {WhatsApp, Facebook} and *Antennas*: {ANT001, ANT004} can be aggregated into *Applications_Antennas*: {*Applications*:WhatsApp, *Applications*:Facebook, *Antennas*:ANT001, *Antennas*:ANT004}, as long as feature sets *Applications* and *Antennas* weigh the same in the authentication decision.

- Regarding sets of numerical features in $\mathscr{S}_t$, for each $S_{t,\,i}$ the following set is built

$$R_{t,i} = \{(j, l): 1 \le j \le n,\ 1 \le l \le s_j^{t,i}\}. \tag{4}$$

For example, from $S = \{2, 3, 1\}$, one would build

$R = \{(1, 1), (1, 2), (2, 1), (2, 2), (2, 3), (3, 1)\}$.

The sizes of these sets are also required to be less than or equal to N.

The protocol proceeds as follows, using the modified sample $\mathscr{R}_0 = \{R_{0,j} \mid 1 \le j \le q\}$:

1. The user initializes a set of Bloom filters $B_{\mathscr{R}_0} = \{B_{R_{0,j}},\ 1 \le j \le q\}$, each of them of size m, by setting all their bits to 0.
2. Then, the device generates a large random number, say key, and stores it in a secure location (for example, a hardware-backed key storage).
3. The user inserts all elements in $R_{0,\,j}$ into $B_{R_{0,j}}$, for $1 \le j \le q$, that is, for every feature $r_l^{0,j} \in R_{0,j}$, the user computes the index set

$$I_{r_l^{0,j}} = \{h_0(r_l^{0,j}, \mathsf{key}), \ldots, h_{k-1}(r_l^{0,j}, \mathsf{key})\}$$

and sets the corresponding bits to 1.
4. Finally, the user sends $B_{\mathscr{R}_0}$ to the server in a confidential manner and deletes $\mathscr{S}_0$, $\mathscr{R}_0$ and $B_{\mathscr{R}_0}$ from the device.

In the last step of the set-up protocol, the user can send $B_{\mathscr{R}_0}$ confidentially to the server by encrypting $B_{\mathscr{R}_0}$ under the server's public key. This protocol can be executed several times within a predefined training period, in order to populate the user's profile $\mathbb{P}$ with additional samples $\mathscr{S}_t$. During this phase, the authentication server also sets the authentication decision threshold, as well as the weights for each of the feature samples, as depicted in Section 3. In general, a rather restrictive threshold is preferable, since any false negatives can be resolved by resorting to explicit authentication.

6.2.2. Authentication protocol

To authenticate the user, the server needs to compute the distances between the feature sets included in the new behavior sample provided by the user and the reference profile (which may consist of several behavior samples).

Let the user's new collected sample be a list of feature sets $\mathscr{S}_f$. The user's device preprocesses the sample to obtain the modified sample $\mathscr{R}_f$, in the same way described in the set-up phase, and builds a set of Bloom filters $B_{\mathscr{R}_f}$, also in the same way described in the set-up phase. The authentication protocol proceeds as follows:

1. The user and the server agree on a fresh random secret key K (for example, using Diffie–Hellman key exchange).
2. The user's device sends $B_{\mathscr{R}_f}$ encrypted under K to the server, and deletes $\mathscr{S}_f$, $\mathscr{R}_f$ and $B_{\mathscr{R}_f}$.
3. The server then computes the distances between the protected sample $B_{\mathscr{R}_f}$ and ℓ previously stored samples in the user profile. The feature sets in each of the samples are assumed to be labeled in such a way that only compatible features are compared. The distances are computed as follows:
   - For sets of categorical features, the server computes $|R_{t,\,j} \cap R_{f,\,j}|$ and $|R_{t,\,j} \cup R_{f,\,j}|$, as described in Section 4.2 and using Expression (2), and obtains the Jaccard distance $d_{J_{t,f,j}}(R_{t,j}, R_{f,j})$, for $f - \ell < t < f$ and all i.
   - For sets of numerical features, the server computes $|R_{t,\,j}|$, $|R_{f,\,j}|$ and $|R_{t,\,j} \cap R_{f,\,j}|$, and obtains the distance as

$$d_{t,f,j} = |R_{t,j}| + |R_{f,j}| - 2|R_{t,j} \cap R_{f,j}| \tag{5}$$

because, using Expression (4),

$$|R_{t,j}| + |R_{f,j}| - 2|R_{t,j} \cap R_{f,j}|$$

$$= \sum_{l=1}^{n} (\max\{s_l^{t,j}, s_l^{f,j}\} + \min\{s_l^{t,j}, s_l^{f,j}\}) - 2\sum_{l=1}^{n} \min\{s_l^{t,j}, s_l^{f,j}\}$$

$$= \sum_{l=1}^{n} (\max\{s_l^{t,j}, s_l^{f,j}\} - \min\{s_l^{t,j}, s_l^{f,j}\}) = \sum_{l=1}^{n} |s_l^{t,j} - s_l^{f,j}|.$$

4. By aggregating the distances for all j's according to their weights, the server obtains a vector $\delta = [\delta_{f-\ell-1,f}, \cdots, \delta_{f-1,f}]$ of the distances between the new sample and ℓ past samples in the user profile. This vector is used then to compute a score (*e.g.* the mean and standard deviation of the distances) which is compared to a threshold t to return an authentication decision.
5. On a successful authentication, the server may store $B_{\mathscr{R}_f}$ as additional reference for authentication. On a failed attempt, the service provider will revert to the default explicit authentication mechanism (such as username-password), and will notify the user of a possibly fraudulent authentication attempt.

7. Privacy and security analysis

As introduced in Section 3, a privacy attacker's objective is to learn the behavior of the authenticated users. We have the following two claims related to privacy.

Claim 1 (*Privacy at the user's device*). *Assuming that the device has not been tampered with, and that subsequent executions are performed in a secure mode (e.g. ARM's TrustZone [1]), an attacker with access to a user's device cannot obtain past feature samples.*

Justification. The user's device deletes the profiles and the protected profiles after set-up and after every authentication attempt. □

Claim 2 (*Privacy at the server*). *The server (or an attacker) does not learn anything about the plaintext profiles other than the number of encoded elements and the sizes of the intersections and unions with other protected profiles of the same user.*

Justification. The server does not receive the plaintext feature sets but Bloom filter encodings of such sets. To recover one feature in the plaintext profile, the server, or any attacker with access to the protected profile, needs to find not just a pre-image of the Bloom filter(s) (see Section 4.3), but *the right* pre-image corresponding to the feature. If the server finds a spurious pre-image (not corresponding to any element), privacy is not violated.

Now, as described in Section 4.4, our instantiation of Bloom filters uses keyed cryptographic hash functions, which, ideally, are resistant to pre-image attacks. These hash functions have much larger domains (as their input includes a long secret key only known to the user's device) and take longer to compute than plain hash functions. This thwarts brute-force attacks attempting to find pre-images by exhaustive search. Hence, finding pre-images (even spurious ones) is not feasible, let alone finding the right pre-images corresponding to inserted features. □

Regarding security against impersonation, we can justify the following claim.

Claim 3 (*Security against impersonation*). *An impersonator with access to the user's device has no better chance to cheat the system than guessing a sample profile close enough to the reference profile stored at the server,*

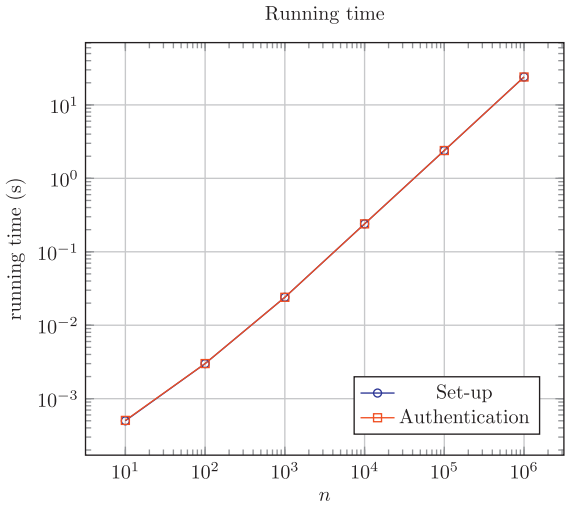

Fig. 5. Categorical features. Running times for different values of the feature set size n. The user profile is assumed to contain a single feature set.

where the impersonator's guess must be made without knowing the reference profile or any previous fresh sample of the legitimate user.

Justification. By Claim 1, having access to the user's device does not give the impersonator access to past samples. Let us examine her other options, which consist of attacking the set-up protocol, the authentication protocol or the Bloom filters used in either protocol.

Attacks during set-up. Since the Bloom filters corresponding to samples in the user's profile are registered in a confidential manner at set-up time (see last step of the set-up protocol in Section 6.2.1), the impersonator cannot get hold of any Bloom filter registered by a legitimate user. On the other hand, the set-up protocol cannot be executed a second time after the predefined training period has expired, and so, an attacker cannot replace after set-up the Bloom filters in the user's profile with new Bloom filters of her choice (the only Bloom filters that can be added to the user's profile after set-up are those corresponding to fresh samples *that were successfully authenticated*, see last step of the authentication protocol in Section 6.2.2).

Attacks during authentication. Each fresh sample is encrypted under a different fresh random secret key K agreed upon by the user and the server in the first step of the authentication protocol (see Section 6.2.2). Hence:

- The attacker cannot get hold of any previous plaintext Bloom filter submitted by the legitimate user, and therefore she cannot re-submit this Bloom filter under a fresh key agreed upon between the attacker and the server.
- The attacker cannot replay any eavesdropped encryption of a Bloom filter under a previous K' agreed upon between the legitimate user and the server, because a fresh K must be agreed upon each time the implicit authentication protocol is run.

Attacks to Bloom filters. Finding a second pre-image for unknown legitimate (B_S, S), where S is a feature set and B_S is a Bloom filter containing the elements of S, is not easier than finding a second pre-image for known (B_S, S); the latter is difficult in our instantiation of Bloom filters based on keyed cryptographic hash functions. Similarly, finding collisions S, S' for unknown legitimate (B_S, S) is not easier than finding collisions for known (B_S, S); the latter is difficult in our instantiation of Bloom filters based on keyed cryptographic hash functions. Hence, the only strategy to pass the authentication protocol that remains to an attacker having access to the user's device is to *guess* a fresh sample $\mathscr{S}_f$ such that its corresponding Bloom filters $B_{\mathscr{S}_f}$ are close to some of the Bloom filters stored by the server for the user to be impersonated. □

8. Experimental analysis

To test the applicability of our mechanism, we implemented our implicit authentication protocol in Python. Our choice for the hash functions in the Bloom filters is $h_i(x, \mathsf{key}) = \text{SHA-512}(x) + i\text{HMAC}(x, \mathsf{key}) \bmod m$, with i ranging from 1 to k. This construction follows the guidelines of [15] and is assumed to be resistant against pre-image and second pre-image attacks; however, new security requirements and attacks may require updating the above choice of hash functions (this cautionary note is similar to the caveats about hash functions in the context of digital signatures). The test bed in which the tests were performed is an Ubuntu x64 14.04 LTS running on an Intel i7-2600 at 3.4 GHz and with 16GB DDR3 1333 MHz.

The proposed mechanism is to be executed in part by the users' devices, and so we conducted an experiment to find out the difference in execution time of hash functions in the test bed and conditions described above against a smartphone. This test consisted in the execution of 100,000 hashes on 64 byte long messages. We used a Samsung Galaxy S7 for comparison. The mean execution time of hash functions in the test bed described above is 6.02 μs, while the execution time in the smartphone is 13.36 μs. This gives a relative difference of 2.21 times. This difference, while not excessive, should be taken into account in all the following tests.

8.1. Speed test for categorical features

In a first test, we checked the speed of the set-up and implicit authentication protocols for different values of n, the size of the feature sets. Parameters m and k were set as per Eq. (3). A user's profile containing a single feature set was considered and the results are shown in Fig. 5. The running time of the set-up protocol does not include the encryption of B_R. Also, the running time of the implicit authentication protocol does not include the execution of the protocol to agree on K. The reason to exclude such cryptographic components is that we use them as a black box: they are not part of the core of our proposal and their running time depends very much on the precise cryptographic algorithms and implementations used.

It may be surprising that the running times of both protocols are so similar. The explanation is that both protocols perform very similar tasks. As described in the previous section, the set-up protocol builds a Bloom filter containing the features in the set and sends it to the server. On the other hand, the implicit authentication protocol builds a Bloom filter with the features in a fresh sample and sends it to the server, who compares it with the reference profile received at set-up time. It turns out that the time needed to compare two Bloom filters is negligible (less than 1 ms) with respect to the time needed to build a Bloom filter (23.4 s for $n = 1,000,000$), so that the latter dominates the total running time of the implicit authentication protocol.

8.2. Accuracy test for categorical features

In this second test, we measured the loss of accuracy introduced by our mechanism in the case of independent categorical features. For such a purpose, we generated 5000 pairs of feature sets of size $n = 50$ with independent categorical features. The first feature set of each pair was taken as the user's profile submitted at set-up time. The second one was taken as a fresh sample submitted at authentication time. Each of the fresh samples was modified by randomly changing up to 50% of their features. Next, with a threshold $t = 0.3$, we classified each of the pairs by computing the Jaccard distance of the pair and tagging it as *accepted* if its distance was below the threshold or *rejected* otherwise. Then, we ran our protocol for all pairs with different values of m and k and we counted how many pairs were misclassified (that is, *accepted* pairs that did not pass authentication, false negatives, plus *rejected* pairs that passed it, false positives). The results are shown in Fig. 6.

The vertical dashed line at approximately $m = 2^{9.5}$ is the optimal

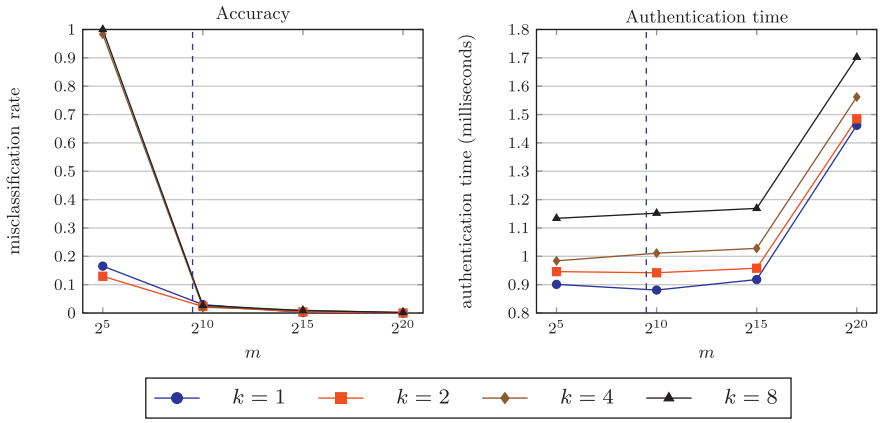

Fig. 6. Categorical features. Accuracy and running times for different values of m and k. Results were obtained as the average of 5000 pairs of feature sets, each containing $n = 50$ features.

value for m as per Expression (3), using $N = n = 50$ and $\rho = 0.001$. We can see that for values of m greater than or equal to the optimal value, the error rate falls below 5%; in fact, it approaches 0% for m close to 2^{20}. Note that for $m = 2^5$ and $k = 4$ or $k = 8$, the error rate is 1. This is because for values of m below the optimal value, the Bloom filter is quickly filled with 1's. When all its m bits become 1's, by Expression (2) the Bloom filter contains ∞ elements ($-\ln 0$); in other words, any element passes the membership test. Hence, a large filter size m is recommended, and the implicit authentication time plot shows that sizes as large as $m = 2^{20}$ are quite manageable in terms of time. Regarding storage and communication, large m values are not problematic either, because Bloom filters can be easily stored and sent in lossless compressed form.

8.3. Accuracy and speed test for numerical features

In this test, we checked the accuracy of our mechanism in the case of independent numerical features. We followed an approach similar to the one in the previous test. We generated a reference profile consisting of a single feature set S_0 consisting of $n = 50$ percentage values adding to 100, that is, a normalized histogram. The features in S_0 might represent the user's 50 most visited websites with their relative frequencies. From S_0, we generated additional feature sets S_i for $1 \leq i \leq 100$, by modifying each of the 50 features of S_0 randomly and re-normalizing. Feature set S_0 was taken as the user's profile submitted as set-up time. Sets $S_1, \ldots, S_{100}$ were taken as fresh samples.

We computed the difference between two samples as

$$d(S_0, S_i) = \sum_{j=1}^{50} |s_j^0 - s_j^i|.$$

The average introduced distance $d(S_0, S_i)$ for $1 \leq i \leq 100$ was 5.43. As our method cannot deal with real numbers, we made a transformation by multiplying by 100 and rounding each feature values. This procedure introduced an error of 0.46%. Note that multiplying feature values by 100 to eliminate the decimal positions increases the size of the sets R_0 and R_i, respectively built during set-up and authentication according to Expression (4), because the maximum values for s_j^0 and s_j^i increase by a factor of 100.

We set $m = 2^{20}$ and $k = 4$. By using our implicit authentication method, the average error introduced for $1 \leq i \leq 100$ was 0.83%.

Beyond measuring the accuracy, we also measured the time taken by the set-up and the implicit authentication protocols. Since we are in the independent numerical feature case, the set-up protocol consisted of building the set R_0 from the user's profile feature set S_0, and building the corresponding Bloom filter (see Section 6.2.1). The mean time in this case was 133 ms. The implicit authentication protocol consisted of building the set R_i from a fresh sample profile S_i, for some $i \in \{1, \ldots, 100\}$, then computing the sizes $|R_0|$, $|R_i|$ and $|R_0 \cap R_i|$ and finally computing the distance from these sizes as per Expression (5). The mean time of the authentication protocol was 133.3 ms (very similar to the mean time of the set-up protocol).

8.4. Tests on the GCU dataset

Finally, we tested our authentication mechanism with the GCU (Glasgow Caledonian University) Dataset Version 1 [13]. The GCU Dataset contains sensor data from 7 Android users. The data consist of WiFi networks, cell towers, application use and light and sound levels.

In this experiment, we used the application usage data to authenticate users. Since these data are categorical and independent, we used the Jaccard similarity coefficient to test the users. The experiment began by choosing one of the 7 users at random, and building a reference user profile by storing 50 samples of the chosen user. Then, we made 1,000,000 authentication attempts, by choosing sample readings at random from all 7 users. We authenticated a user when the average similarity, computed as the average similarity of the random sample against the 50 reference samples, was above a pre-specified threshold, taking into account the standard error of the computed average.

In Fig. 7, we show the performance metrics of this authentication mechanism for the GCU dataset, including false positive and negative rates, for increasing values of the threshold. Fig. 7(a) shows the results of the experiment when computing the distances with cleartext samples, while Fig. 7(b) shows the results for comparisons using our proposed mechanism based on Bloom filters. Note that the results of both approaches are nearly identical, showing that the use of Bloom filters to protect the user profiles does not significantly affect the authentication accuracy with respect to comparing the profiles in the clear.

An additional positive side effect of using Bloom filters is that, because of their nature, that is, being bit strings of a fixed length, the storage requirements for such an authentication mechanism become simpler to predict than those of a mechanism which uses the raw data, which are sets of variable size of numerical and/or categorical values.

8.5. Comparison with previous proposals

Privacy-preserving implicit authentication requires an underlying private matching scheme. The major private matching schemes in the literature fall into two categories: those based on garbled circuits and the Freedman–Nissim–Pinkas scheme [8].

The existing privacy-preserving implicit authentication schemes [6,18] are based on [8]. The oldest one, [18], requires the user to compute two encryptions (one using Paillier and the second one using an order preserving encryption scheme) for each feature, then decrypt a Paillier encryption and reencrypt using the OPSE scheme. On the other

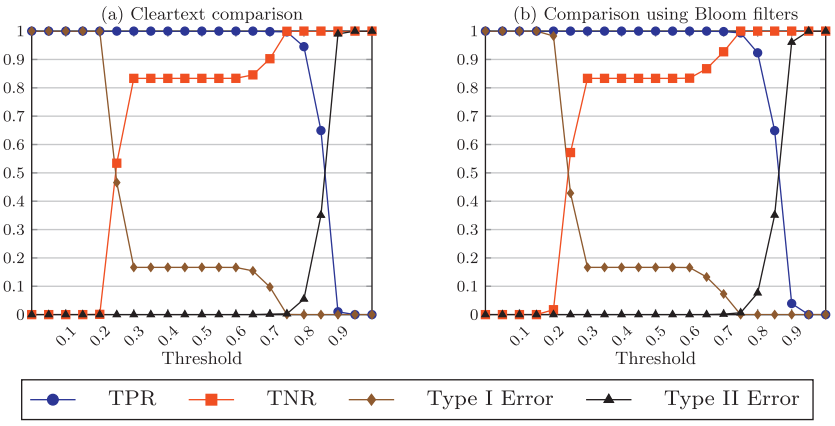

Fig. 7. Performance indicators of the implicit authentication mechanism on the GCU dataset. TPR stands for true positive rate (correct authentication) and TNR for true negative rate (correct non-authentication). Type I Error stands for false positive rate and Type II for false negative rate.

hand, our previous scheme [6] requires computing only one Paillier encryption per feature, so it outperforms [18] by being able to perform tens of comparisons in tens of seconds.

If one resorts to garbled circuits as a private matching scheme, the current implementations of this generic multiparty construction allow comparing up to ten thousand set elements in the order of hundreds of seconds [10].

In contrast, our current proposal allows comparing feature sets of 1 million elements in the order of tens of seconds.

This radical performance improvement over previous works comes at the expense of some relaxation in security. Whereas the homomorphic encryption used in our previous papers was resistant to malicious user devices, the approach based on Bloom filters presented in this paper requires the user device to be protected against malware. All in all, though, the fact that the new approach allows basing authentication on more comprehensive sets of features compensates the previous shortcoming: by protecting the user device against malware, one attains a much more accurate implicit authentication.

9. Conclusions and future work

We have proposed a computationally efficient privacy-preserving implicit authentication protocol. Our protocol builds on the work in [6], but avoids the high complexity of that proposal that limited the size of user profiles that could be managed. To make the computation lighter, we have used the properties of Bloom filters to calculate the sizes of the union and intersection of encoded sets. Our protocol is simple and fast, and therefore ready to be implemented in production systems. Additionally, the privacy of the user profiles is protected because the profiles cannot be recovered from their Bloom filter encodings.

Our efficiency improvement, however, comes at the cost of losing the semantic security provided by the aforementioned protocol. In an extreme scenario, such a loss might impact on the privacy of our solution. We plan to solve this problem in future work, by considering the use of oblivious transfer protocols and homomorphic encryption (such as Goldwasser–Micali).

Another line of future research relates to finding ways of using Bloom filters to deal with correlated features in profiles, that is, features that are not independent of each other (for example, if the feature values are the IDs of cell towers or Internet access points seen by the device, nearby cell towers/access points are more similar to each other than distant cell towers/access points).

Acknowledgments and disclaimer

The following funding sources are gratefully acknowledged: European Commission (projects H2020 644024 “CLARUS” and H2020 700540 “CANVAS”), Government of Catalonia (ICREA Acadèmia Prize to J. Domingo-Ferrer and grant 2014 SGR 537), Spanish Ministry of Economy, Industry and Competitivity (project TIN2014-57364-C2-R “SmartGlacis”). The authors are with the UNESCO Chair in Data Privacy, but the views in this paper are their own and do not necessarily reflect those of UNESCO.

References

[1] ARM TrustZone, https://www.arm.com/products/security-on-arm/trustzone. Accessed.
[2] A. Blanco-Justicia, J. Domingo-Ferrer, O. Farràs, D. Sánchez, Distance computation between two private preference functions, Proceedings of the 29th IFIP International Information Security Conference, Springer, 2014, pp. 460–470.
[3] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[4] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, Y. Tang, On the false-positive rate of bloom filters, Inf. Process. Lett. 108 (4) (2008) 210–213.
[5] N.L. Clarke, S.M. Furnell, Authenticating mobile phone users using keystroke analysis, Int. J. Inf. Secur. 6 (1) (2007) 1–14.
[6] J. Domingo-Ferrer, Q. Wu, A. Blanco-Justicia, Flexible and robust privacy-preserving implicit authentication, Proceedings of the 30th IFIP International Information Security Conference, Springer, 2015, pp. 18–34.
[7] C. Dong, L. Chen, Z. Wen, When private set intersection meets big data: an efficient and scalable protocol, Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, 2013, pp. 789–800.
[8] M.J. Freedman, K. Nissim, B. Pinkas, Efficient private matching and set intersection, Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques-EUROCRYPT, Springer, 2004, pp. 1–19.
[9] T. Gerbet, A. Kumar, C. Lauradoux, The power of evil choices in Bloom filters, 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (2015) 101–112. IEEE.
[10] Y. Huang, D. Evans, J. Katz, Private set intersection: are garbled circuits better than custom protocols? NDSS, (2012).
[11] M. Jakobsson, E. Shi, P. Golle, R. Chow, Implicit authentication for mobile devices, Proceedings of the 4th USENIX conference on Hot topics in security, USENIX Association, 2009. 9–9.
[12] M. Kantarcioglu, R. Nix, J. Vaidya, An efficient approximate protocol for privacy-preserving association rule mining, Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2009, pp. 515–524.
[13] H.G. Kayacik, M. Just, L. Baillie, D. Aspinall and N. Micallef, Data driven authentication: on the effectiveness of user behaviour modelling with mobile device sensors, arXiv:1410.7743.
[14] F. Kerschbaum, Outsourced private set intersection using homomorphic encryption, Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ACM, 2012, pp. 85–86.
[15] A. Kirsch, M. Mitzenmacher, Less hashing, same performance: building a better bloom filter, Random Struct. Algorithm. 33 (2) (2008) 187–218.
[16] J. Muncaster, M. Turk, Continuous multimodal authentication using dynamic Bayesian networks, Proceedings of 2nd Workshop Multimodal User Authentication, Toulouse, France, 2006.
[17] B. Pinkas, T. Schneider, M. Zohner, Faster private set intersection based on OTextension, 23rd USENIX Security Symposium (USENIX Security 14), (2014), pp. 797–812.
[18] N.A. Safa, R. Safavi-Naini, S.F. Shahandashti, Privacy-preserving implicit authentication, Proceedings of the IFIP International Information Security Conference,

Springer, 2014, pp. 471–484.
[19] W. Shi, J. Yang, Y. Jiang, F. Yang, Y. Xiong, SenGuard: passive user identification on smartphones using multiple sensors, Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference, IEEE, 2011, pp. 141–148.
[20] R. Schnell, T. Bachteler, J. Reiher, Privacy-preserving record linkage using Bloom filters, BMC Med. Inform. Decis. Mak. 9 (1) (2009) 41.
[21] S.J. Swamidass, P. Baldi, Mathematical correction for fingerprint similarity measures to improve chemical retrieval, J. Chem. Inf. Model 47 (3) (2007) 952–964.
[22] S. Taheri, M.M. Islam, R. Safavi-Naini, Privacy-enhanced profile-based authentication using sparse random projection, IFIP International Conference on ICT Systems Security and Privacy Protection, Springer, Cham, 2017, pp. 474–490.