Anonymous Authentication With Pseudonyms

(Extended Abstract of the MSc Dissertation)

GUILHERME SANTOS, Instituto Superior Técnico, Universidade de Lisboa, Portugal
SUPERVISOR: PROFESSOR LUÍS RODRIGUES, Instituto Superior Técnico, Universidade de Lisboa, Portugal

The problem of anonymous authentication, which is key to preserve the privacy of users, has an increased relevance given the proliferation of applications that use components deployed in the network, that cannot be fully trusted. A possible strategy to preserve user privacy is to implement authentication using pseudonyms. In this thesis, we survey how pseudonyms have been used to preserve privacy in different application areas and discuss the advantages and limitations of the different approaches to manage pseudonyms. Based on the limitations of previous work, we design, implement and evaluate novel techniques to enhance the performance of systems that use pseudonyms.

Additional Key Words and Phrases: Privacy, Anonymous Authentication, Pseudonyms

1 Introduction

Client authentication is a requirement for many applications, as it is common for certain resources or functionalities to be accessible only to authorized users. Unfortunately, the need for authentication creates opportunities for the collection of data related to clients' usage profiles, which can be used against their interests (for example, to inflate the price of a service based on user habits).

One way to preserve user privacy in applications where authentication occurs frequently is through what is known as anonymous authentication, where customers authenticate using different pseudonyms, every time they access the system.

If a client violates the service usage rules, it may be necessary to prevent them from accessing the application, which requires the revocation of previously issued pseudonyms. One of the challenges of anonymous authentication is to preserve client privacy during the revocation process. In particular, the fact that two pseudonyms appear on a revoked pseudonym list may indicate that they belong to the same client, potentially violating the client's privacy, especially if these pseudonyms have already been used or will be used by the client in the future.

In this context, two relevant properties of the authentication process are *backward unlinkability*, which ensures that an adversary cannot determine whether two distinct authentications were performed by the same client, and *revocation auditability*, which allows the client to verify whether a given pseudonym has been revoked before using it. Revocation auditability is typically achieved by transferring the updated revocation list to the client during the authentication process.

Although many proposed systems offer relevant properties, securing the privacy of clients, these type of systems are inefficient, and suffer from high latencies during the authentication process, due to the revocation list transfer, which can take several dozens of seconds, making them unfeasible to be applied in many scenarios.

In this work, we propose and evaluate a new set of tools to improve the performance of anonymous authentication systems. Specifically, we develop techniques that allow a client to verify

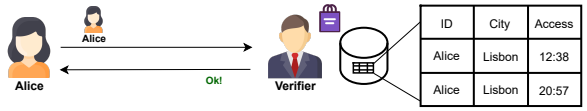


Fig. 1. Customer profiling through the authentication process.

Fig. 2. Linkability problem when revoking a user. RL is Revocation List. P is Pseudonym.

that they have not been revoked without having to download the entire revocation list. Many of these systems implement their revocation lists using Bloom Filters. We propose two different solutions that enable the client to transfer less data: i) a technique based on Redactable Signatures [7], which allows a client to request only specific entries from a Bloom Filter; ii) a technique based on Hierarchical Bloom Filters [12], which enables a client to sequentially download smaller filters instead of the original one.

Both techniques preserve all necessary security properties, such as the integrity and authenticity of the filters, a particularly interesting challenge for the first technique. Our component allows the client to receive only reduced portions of the filters for authentication.

Through the experimental evaluation of our solution, we demonstrate that it reduces the time required to perform authentication, in a specific system, from 17 seconds to approximately 2 milliseconds, considering a revocation list of 1.25GB.

2 Background

In the following paragraphs, we begin by introducing *Bloom filters*, one of the most commonly used data structures in authentication schemes, as well as an extension of them, known as *hierarchical Bloom filters*. Finally, we present *redactable signatures*, which ensure integrity and authenticity over partial messages.

2.1 Bloom Filters

Bloom filters [2] are a probabilistic data structure designed to store information about the membership of objects in a set and efficiently test whether a given object belongs to the set. A Bloom filter consists of a vector of binary digits (bits). To record the presence of an object in the set, multiple hash functions are applied to the object, each function computing a position in the vector that must be set to "1." To test whether an object belongs to the set, the same process is

applied, checking if the bits indicated by the hash functions are set to "1." If they are, the object is considered to be part of the set.

Since Bloom filters are a probabilistic structure, they allow false positives (but prevent false negatives), meaning that an object may be incorrectly identified as belonging to the set when it does not. The false positive rate depends on several factors, such as the size of the filter, the number of elements in the set, and the number of hash functions used. A larger filter can store more elements while maintaining the same false positive rate.

In pseudonym-based authentication systems, Bloom filters are commonly used to encode the revocation list and track which pseudonyms have been revoked. In large-scale systems, due to the number of pseudonyms to be revoked and the need to keep the false positive rate low, the size of Bloom filters can become quite large, making frequent transmission costly. It is important to note that the false positive rate is chosen *a priori*, based on system requirements, and the filter's characteristics are then determined accordingly.

2.2 Hierarchical Bloom Filters

Hierarchical Bloom filters [12] are a data structure based on Bloom filters that aim to improve the scalability of the original Bloom filters. These filters use a hierarchical structure composed of multiple Bloom filters of different sizes, where filters at higher positions in the hierarchy are smaller than those at lower positions. The insertion and membership testing operations are performed across multiple filters.

Typically, a membership test in this data structure requires performing sequential presence checks in the various filters until a conclusion is reached.

2.3 Redactable Signatures

Redactable signatures [7] enable the sending of a digitally signed message, while removing certain parts of it, maintaining the recipient's ability to verify the security properties of the message, such as integrity and authenticity. The process of creating a redactable signature starts by dividing the message into removable parts. A Merkle tree is constructed, associating each part of the message with each leaf node of the tree by calculating its cryptographic digest, then recursively building the tree up to the root node, where the value of each node is the digest of the concatenation of its children's digests. After this, the message sender digitally signs the root node.

When sending the message, the sender removes the parts of the message they wish to exclude and sends the remaining parts to the recipient. In addition to these, they also send the digests of the leaf nodes associated with the removed parts, which can be compressed using the tree structure by sending the digests of the internal nodes. The recipient uses the information received to reconstruct the digest of the root node and verify the sender's digital signature on that node.

Redactable signatures also use a random component during the signature creation, which prevents the client from recovering the removed parts through brute-force attacks.

3 Related Work

Several anonymous authentication systems have been proposed based on pseudonyms.

We start by explaining the general structure that almost every pseudonym system follows. Generally, this kind of systems are usually composed by 3 entities: 1) the clients, 2) the verifiers that control the authentication process and, 3) a central authority trusted by both clients and verifiers which is a pseudonym provider, that provides clients with enough pseudonyms for their authentications, keeps a map between users' real identities and the pseudonyms issued, and handles clients' revocations, distributing the revoked pseudonyms to the system verifiers using revocation lists, as shown in Figure 3.

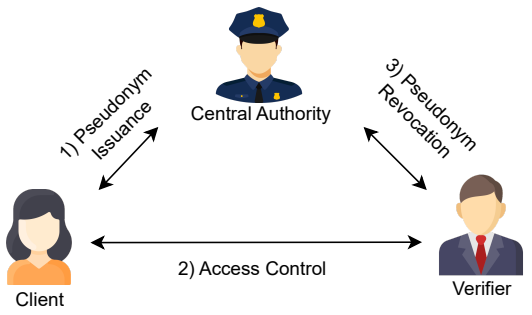


Fig. 3. Anonymous Authentication Systems Overview.

The pseudonyms issued have normally an asymmetric key pair associated and are made up of the public key and the pseudonym provider's signature. When a client is authenticating in the system, he uses a pseudonym as his digital identity and sends a different one to the system verifier in each authentication. The purpose of the pseudonym's signature is to prove to the verifier the authenticity and integrity of the pseudonym showing that the pseudonym was, indeed, created by the pseudonym provider. This scheme takes advantage of the asymmetric encryption properties to prove that the client is the real owner of a given pseudonym, by using the associated private key. So, the messages exchanged between the system verifier and a client are usually appended with the pseudonym and a signature made by the client using the private key associated with that pseudonym.

These systems also follow similar steps, as shown in Figure 3. These steps are usually divided in 3 phases: 1) Pseudonym Issuance, 2) Access Control and, 3) Pseudonym Revocation. Whenever a client joins the system, must start by asking the pseudonym provider for pseudonyms for his authentications. After acquiring his pseudonyms, a client is ready to authenticate in the system with a verifier. The authentication is divided in three phases: the verifier sends to the client, information about his revocation status, the client checks this information and if he is not revoked, sends a pseudonym to the verifier, the verifier checks the validity and authenticity of this pseudonym, and if everything is correct the user

accesses the system. If an authenticated client misbehaves, the system generates a complaint, inserting the pseudonym the client used, and sends it to the pseudonym provider. The pseudonym provider links the pseudonym inserted in the complaint to the real identity of a client and then revokes all the pseudonyms issued before to that client and distributes them in a revocation list to the system verifiers.

This is just a general approach followed by most systems, but some other systems may present some radical approaches or simply do not implement some steps of this approach.

Backward Unlinkability. Some pseudonym systems [5, 6, 10, 11], aim for backward unlinkability by dividing time into large epochs, each further split into smaller time slots. Clients receive pseudonym groups tied to specific slots, which cannot be used outside their assigned time. Upon revocation, only pseudonyms for future slots are added to the revocation list and shared with verifiers. This approach doesn't ensure perfect backward unlinkability—either revealing past authentications during a slot or allowing, for a revoked user, continued access until the slot ends. This issue can be mitigated by reducing slot size, though it increases the number of pseudonyms linearly. Thus, these systems must balance slot size with pseudonym overhead.

Privacy Keeper [4] avoids the use of time slots by employing non-revocation proofs. In this approach, customers present a pseudonym along with a proof—specific to the current revocation list—demonstrating that they have not been revoked. When a client is revoked, a new revocation list is generated, and non-revocation proofs for all previously revoked pseudonyms are computed and added to the list. ince each proof is unique to its corresponding revocation list, verifiers cannot link past authentications to the current list, enhancing unlinkability.

Revocation Auditability. To provide revocation auditability is necessary that the client is able to know his revocation status before authentication. Nymble [10] and Privacy Keeper [4] accomplish this by having the verifier sending the revocation list in the beginning of each authentication, so the client can effectively check if his pseudonyms are recvoked or not. The fact of transferring the revocation list in every authentication process, makes the task of providing revocation auditability onerous, and a very slow process that can reach serveral seconds. making it not practical.

Revocation Lists. Revocation lists are essential in pseudonym systems that rely on a trusted third party to mediate trust between clients and verifiers, ensuring that pseudonyms assigned to revoked users are properly flagged. To reduce the overhead of distributing and storing these lists, many systems implement them using Bloom filters—a space-efficient probabilistic data structure. The design of the pseudonym system significantly affects both the frequency and size of the revocation lists. In time slot–based approaches, a new revocation list can be created for each time slot. When a user is revoked, only the pseudonyms associated with the relevant time slot are added to that slot's revocation list. This method requires frequent, but leads to the creation of shorter revocation lists. On the other hand, in systems like Privacy Keeper, when generating a revocation list, they include a non-revocation proof for every pseudonym of every revoked user. This results in significantly larger revocation lists.

Zero-knowledge proofs. Some different systems, such as Perea [9], do not require the presence of a trusted third party. They employ the use of zero-knowledge proofs, during authentication process, between verifiers and clients. Revocation lists are based on cryptographic accumulators [3, 8]. In the initial phase, clients get pseudonyms from verifiers. During the authentication phase, the revocation list is sent to the client which confirms his revocation status. With that information, the client decides whether to authenticate in the system or not. If so, the client authenticates, by generating a non-revocation proof, not disclosing those pseudonyms, proving that none of the last pseudonyms used are in the revocation list. Although these systems provide perfect backward unlinkability and perfect revocation auditability, zero-knowledge proofs impose high latencies on the authentication process.

4 Proposed Techniques

A disadvantage of pseudonym systems that provide the auditability property of revocation is that they require the transfer of the revocation list to the client, which, given its potentially large size for very dynamic systems, can introduce significant latency in the authentication process. Our work focuses on this problem, proposing techniques that aim to reduce the amount of information that clients and verifiers exchange during each authentication process, while maintaining all the security properties inherent to the revocation list, namely its integrity and authenticity. Considering that most of the pseudonym systems use Bloom filters to implement the revocation list, we propose two techniques to achieve this: 1) a technique based on Hierarchical Bloom Filter Arrays, and 2) a technique inspired by Redactable Signatures. In both techniques proposed in this work, we assume the same entities as in most systems discussed: clients, verifiers, and the central authority, as well as the same phases: pseudonym issuance, pseudonym revocation, and access control, with substantial changes in the following phases: pseudonym revocation and access control.

From this point forward, it is used BF to refer to Bloom Filter and RL to refer to Revocation List.

4.1 Technique based on Hierarchical Bloom Filter Arrays

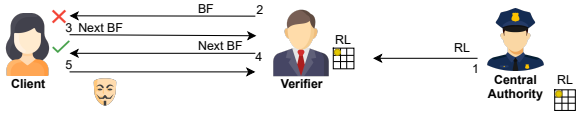


Fig. 4. Authentication with Hierarchical Bloom Filter Arrays. BF is Bloom Filter. RL is Revocation List.

In this technique, we propose that the RL be composed of multiple BF of different sizes, in addition to the original filter, each containing exactly the same elements. These filters are organized sequentially and in ascending order of their size.

4.1.1 Pseudonym Revocation. During the revocation of a client, the certification authority creates a set of n new BF FBn_η to form the new RL. All the elements used to revoke the pseudonyms to be inserted in that list, being the elements the pseudonyms themselves or non-revocation proofs constructed for each revoked pseudonym, are inserted into all the BF created. Finally, the central authority signs each of these filters with its private key, protecting the authenticity and integrity of each filter.

Subsequently, these filters and their respective digital signatures are aggregated into a RL, $RL_{new} = \langle [BF1, ..., BFn], [BF1^{K^-_{CA}}, ..., BFn^{K^-_{CA}}] \rangle$, which is sent to the verifiers.

4.1.2 Access Control. During the authentication process, represented in Figure 4, the client starts by downloading the first BF from the RL, which is the one with the smallest size, as well as, other relevant information for the pseudonym system, such as the random parameter η associated with the RL. Then, they choose a pseudonym p and checks if this pseudonym is marked as revoked in the previously received filter. Since the false positive rate of a BF varies with its size, and the client begins by receiving the smallest filter, it is not unlikely that the client encounters a false positive at this step. In that case, the client will then download the remaining filters from the RL in increasing order of size until they reach a filter where the pseudonym is not marked as revoked. If they reach the last filter and the pseudonym is marked as revoked in that filter, the client assumes that it has indeed been revoked and cancels the authentication. The authenticity of each filter is verified by the client using the certification authority's signature. If the client concludes that they have not been revoked, they can then complete the authentication by sending the pseudonym p and the corresponding proof to the verifier.

This technique is advantageous in cases where the client can confirm that the pseudonym has not been revoked by checking the first few filters (ideally, relying only on the smallest filter in most cases); however, it may worsen the original solution if the client has to download several filters. In most cases, the client would be able to authenticate using the first pseudonyms by checking only the smaller filters, making this an efficient solution overall.

4.2 Technique based on Redactable Signatures

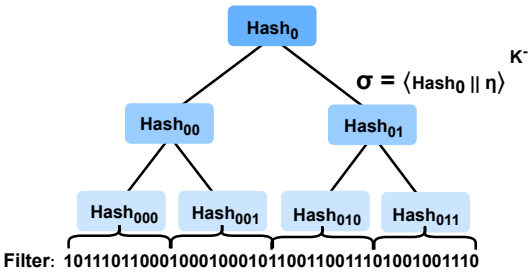


Fig. 5. Redactable Signature Creation.

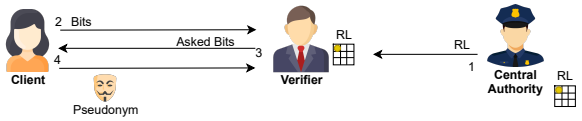


Fig. 6. Authentication with Redactable Signatures. RL is Revocation List.

In this technique, we take advantage of the fact that the client can know in advance the characteristics of the Bloom filter encoding the revocation list, such as its size, the number of hash functions. Using this information, the client can calculate their non-revocation proof and determine which entries in the vector they need to check to see if a given pseudonym has been revoked. We then propose that the client, instead of downloading the entire filter, only downloads segments of the filter in order to obtain all the necessary entries from the vector.

4.2.1 Pseudonym Revocation. By transferring only parts of the revocation list, the client is not able to verify the integrity of the list using a standard digital signature, as proposed in Privacy Keeper. To overcome this issue, we suggest that the central entity signs the revocation list using a technique inspired by redactable signatures.

In our solution, after creating the revocation list, $RL_{new} = \langle BF \rangle$, the Bloom filter is divided into multiple segments of configurable size, and the cryptographic hash of each segment is associated with the child nodes of a Merkle tree. The remaining nodes of the tree are then generated recursively by hashing the concatenation of the hashes of the child nodes. Finally, the central authority generates a digital signature σ, by concatenating the root hash with the parameter η associated with the revocation list (unique value in Privacy Keeper or time slot in Nymble), as illustrated in Figure 5. The generated elements form the new revocation list, $RLnew = \langle BF, \sigma \rangle$, which is distributed to the verifiers, who then reconstruct the Merkle tree using the same process.

4.2.2 Access Control. In the access control phase, shown in Figure 6, the verifier begins by sending the information about the revocation list, such as: the characteristics of the Bloom filter, or the unique value η in the case of the anonymous system, Privacy Keeper.

With this information, the client selects a pseudonym p that they have not used yet and with that pseudonym p, calculates the exact positions of the Bloom Filter that needs to verify to test their revocation status, (in the case of the system Privacy Keeper, the client would first need to calculate the non-revocation proof), and requests these positions. The verifier responds by sending the filter segments that have the bits requested along with the necessary hashes for the client to reconstruct the path from the leaf nodes associated with the requested segments to the root node.

Upon receiving this information, the client ensures they are not revoked using the Bloom filter segments they received, checks the signature σ from the certifying entity, and then calculates the root hash recursively using the information sent by the verifier. The integrity of the Bloom filter segments received by the client is safeguarded by the structure of the binary tree. If a malicious verifier were to modify any part, that change would reflect in some higher

node of the tree due to the properties of the Merkle tree. The alteration would necessarily affect the root hash, allowing the client to notice the change when verifying the signature from the certifying authority at that same node. If the client concludes that they are not revoked, they complete the authentication process by sending their pseudonym p and the corresponding proof to the verifier.

We propose an optimization to this technique specifically for cases where the client attempting to authenticate is not revoked, further reducing the amount of information propagated through the network. For non-revoked clients, it is sufficient to ensure that at least one bit of the bits needed to verify their pseudonym's validity, is set to zero in the Bloom filter. Accordingly, we introduce this optimization, having the verifier send only a single segment of the Bloom filter containing a bit requested by the client, that is set to "0", rather than transmitting all segments corresponding to the client's requested bits.

It is important to note that this optimization does not apply to revoked clients. Revoked clients must receive all the bits required to validate their pseudonym and confirm that each bit is set to "1", thereby verifying their revoked status.

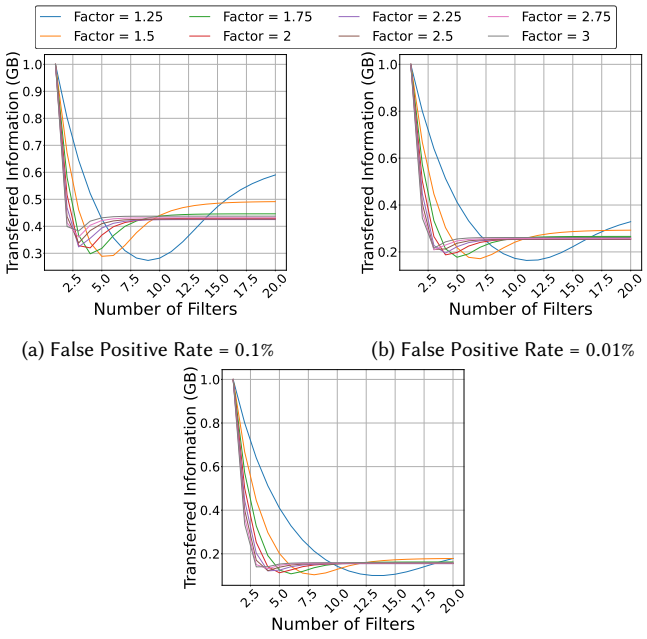
5 Evaluation

We experimentally evaluated both techniques proposed by implementing each technique in C++ for two different anonymous authentication systems: Privacy Keeper and Nymble. For each system, we developed a client and a verifier using two Intel NUC10i7FNB machines. The machines are equipped with an Intel i7-10710U processor, 16 GB of RAM, and run Ubuntu 22.04 LTS. We analyzed the impact of varying different parameters of both solutions on the amount of information transmitted during authentication for both systems. We also evaluated both techniques by measuring the average authentication latency and comparing it with the average latency of Privacy Keeper. We used the Ed25519 algorithm [1] to generate deterministic digital signatures.

5.1 Technique based in Hierarchical Bloom Filters

As previously described, our technique based on hierarchical Bloom filters uses multiple filters of different sizes, where various parameters directly influence the amount of information that needs to be transferred during authentication. Thus, we evaluate the following parameters: 1) *reduction factor* between each filter, starting from the largest/original filter down to the smallest one. 2) *number of filters* with different sizes used in the hierarchy, meaning how many times we reduce the largest filter by the chosen factor. 3) *false positive rate* that we accept in the largest filter of the hierarchy. In Figure 7, we vary these parameters and compute the expected amount of Bloom filter-related information that needs to be transferred at authentication time for each configuration, fixing the size of the largest filter at 1GB.

It is worth highlighting that computing the expected amount of transferred information is not trivial since it depends on the number of filters the client has to transfer before authenticating. To evaluate each set of parameters, we calculated the amount of transferred information for each scenario (in each scenario, the client needs to download a different number of filters before completing the audit process) and computed the expected value of the information transferred by the client using the false positive rates of each Bloom filter to determine the probability of each scenario.



(a) False Positive Rate = 0.1%

(b) False Positive Rate = 0.01%

(c) False Positive Rate = 0.001%

Fig. 7. Variation of Parameters using Hierarchical Bloom Filters Arrays

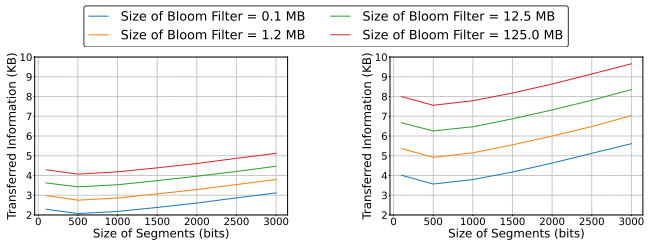
It can be observed that when there is only one filter, the required amount of transferred information is always 1GB, representing the original filter. However, when more filters are created, the average amount of transmitted information quickly decreases to less than half when the number of filters exceeds 5. Moreover, we observe that the smaller the reduction factor, the lower the expected amount of transmitted information in each authentication, although it becomes necessary to increase the number of filters used to achieve the minimum transmitted information.

We also observe that the transferred information decreases when we reduce the false positive rate. The explanation lies in the fact that reducing the rate also affects the rates of the smaller filters, as all filters will have lower false positive rates, leading to a larger number of clients authenticating at the first filters. Note that, to decrease the false positive rate while keeping the filter size fixed, it is necessary to reduce the number of items in the filter.

A good configuration for a false positive rate of 0.001% would be to choose a factor of 2 with 4 filters, as this point minimizes the information transmitted over the network while maintaining a reasonable number of filters. Our technique drastically reduces the transferred information, in particular, transmitting only about 10% of the original information on average.

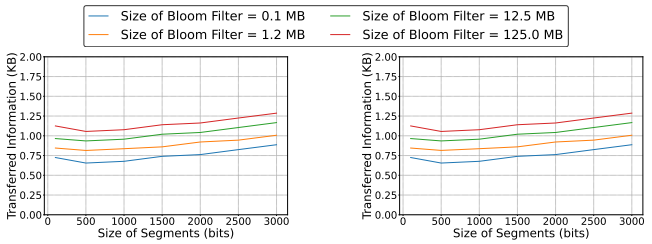
5.2 Technique based in Redactable Signatures

Our technique based on Redactable Signatures has two main structures: the Bloom filter and a Merkle tree, which is built from the



(a) Number of hash functions = 5 (b) Number of hash functions = 10

Fig. 8. Variation of Parameters using Redactable Signatures



(a) Number of hash functions = 5 (b) Number of hash functions = 10

Fig. 9. Variation of Parameters using Redactable Signatures using proposed Optimization

filter. The information transferred during authentication consists mainly of segments of the Bloom filter and multiple digests from the Merkle tree.

Several parameters of the solution impact the amount of information transferred during authentication. These parameters include: 1) *number of hash functions*, which directly influences the number of Bloom filter segments sent to the client, and 2) *segment size*, into which the filter is divided, as its size determines the number of filter segments that will be associated with the leaf nodes of the tree, affecting the size of the Merkle tree and the number of digests sent to the client.

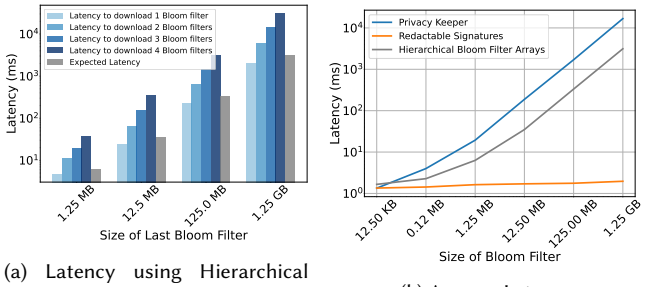
In Figure 8, we vary these parameters and measure the average amount of information that the verifier sends to the client during authentication.

It is noticeable that increasing the number of hash functions in the filter increases the amount of information transmitted over the network. This happens because the client receives more segments as the number of hash functions grows, and also because more digests need to be sent so that the client can recursively compute the paths from the leaf nodes to the root node.

The amount of information transferred during authentication reaches its minimum, considering the studied filter sizes and number of hash functions, when the segment size is 500 binary digits. This can be understood by noting that with shorter segments, the associated Merkle tree becomes larger, requiring more digests to be sent to the client. Conversely, with larger segments, more information about them needs to be sent to the client. The optimal value balancing these two parameters is 500 binary digits.

From this experiment, we conclude that a good configuration would be to divide the Bloom filter into segments of 500 binary digits. In the worst-case scenario, if the filter had 10 hash functions, the solution would reduce the information sent to the client from 125MB to 7KB. It is worth highlighting that this represents a significant reduction in the original amount of information (specifically, 0.006% of the original size).

5.3 Hierarchical Bloom Filters vs Redactable Signatures



(a) Latency using Hierarchical Bloom Filter Arrays (b) Average Latency

Fig. 10. Authentication Latency

In this section, we evaluate the authentication latency of both proposed techniques and compare them with the authentication latency of Privacy Keeper. To ensure a fair comparison between both techniques, we selected configurations that minimize the information exchanged during authentication between verifiers and clients, based on the analysis presented earlier.

For the evaluation of all systems, we developed a set of tests where we measured the average authentication latency of a client while varying the size of the Bloom filter that implements the revocation list. In the case of the technique based on hierarchical Bloom filters, this size corresponds to the size of the last and largest filter. For the evaluation of PickyFilters with hierarchical Bloom filters, we used the following configuration: a reduction factor of 2 (meaning each filter is half the size of the next one), fixing the number of filters in the set to 4, with 5 hash functions per filter. By varying the latency of the last filter, we measured the authentication latency in each of the 4 possible scenarios (in each scenario, the client downloads a different number of filters). Using these 4 latencies, we calculated the expected latency of each authentication, considering the latency for each of the 4 scenarios and the false positive rate of each filter, which we present in Figure 10a. To calculate the expected latency, we assumed a false positive rate of 0.01% for the last Bloom filter, which we used to derive the false positive rates for the other filters. Note that only this technique has its efficiency dependent on the acceptable false positive rate in the revocation list. We used the obtained expected latency for comparison with the other two techniques. For the evaluation of the technique based on Editable Signatures, we fixed the segment size to 500 binary digits and the number of hash functions to 5, measuring the authentication latency.

In Figure 10b, we compare the expected latencies (and their evolution as the Bloom filter size increases) using both proposed techniques and the original Privacy Keeper (which always transfers the

entire revocation list). Our experimental evaluation shows that for Bloom filters of 1.25GB, the average authentication latency is approximately $\pm 17s$ in the original Privacy Keeper, $\pm 3s$ with hierarchical Bloom filters, and $\pm 2ms$ with Redactable Signatures. PickyFilters with Editable Signatures thus achieves the best performance, significantly reducing authentication latency by approximately 99.99% compared to the original Privacy Keeper.

5.4 Evaluation Based on System Scale

In this section, we evaluate the effectiveness of both proposed techniques implementing them in Privacy Keeper and Nymble, by measuring the amount of information the verifier needs to send to the client during the process of authentication, varying the number of revoked users in the systems, and comparing the techniques with the original systems. To ensure a fair comparison between the techniques, we selected configurations that minimize the information exchanged during authentication between verifiers and clients, as outlined in the earlier analysis.

As explained before, the number of users revoked affects the size of the revocation lists differently. In Privacy Keeper, a non-revocation proof for each revoked pseudonym is added to the revocation list. Consequently, the number of elements in the revocation list is equal to the number of revoked users multiplied by the average number of pseudonyms per revoked user. In Nymble, which operates using epochs and slots where each user is assigned a pseudonym for each time slot, revocation lists can be generated at the start of each slot independently. As a result, the number of elements in each revocation list corresponds exactly to the number of revoked users.

For both techniques, we used exactly the same configuration as in the evaluation of section **??**. For the evaluation of the technique with HBFA, we used the following configuration: a reduction factor of 2, which means that each filter is half the size of the next filter, with the number of filters in the set fixed at 4, and 5 hash functions in each filter. For the evaluation of the redactable signatures-based technique, we fixed the segment size at 500 binary digits and the number of hash functions at 5. We consider a false positive rate of 0.01% for revocation lists. We also consider 300 pseudonyms per revoked user in Privacy Keeper.

We calculated the amount of information transferred for the HBFA technique using the same approach described in section **??**. Specifically, we measured the information transferred in each of the four possible scenarios, where the client downloads a varying number of filters in each case. Using these four values, we determined the expected amount of information transferred per authentication by accounting for the information transferred in each scenario and the false positive rate of each filter, which being 0.01% the rate of the last filter.

In Figure 11, we compare both techniques with the original systems. Our study shows that both our proposed techniques are effective for both systems, allowing the systems to scale up without losing performance during the authentication protocol, being absolutely critical for the Privacy Keeper and its applicability, and a huge improvement for Nymble. In particular, in a system with 1 million revoked users, our technique based on redactable signatures,

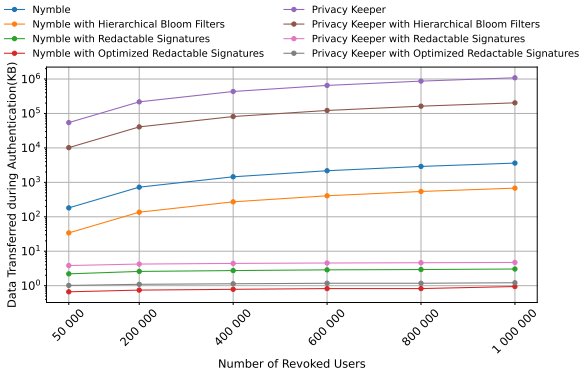


Fig. 11. Data Transferred during Authentication vs Number of Revoked Users

allows to lower the amount of transferred information from 1.09GB to 1.2KB, in the case of Privacy Keeper, and from 3.6MB to 0.9KB, in the case of Nymble.

5.5 Additional Time to Generate a RL

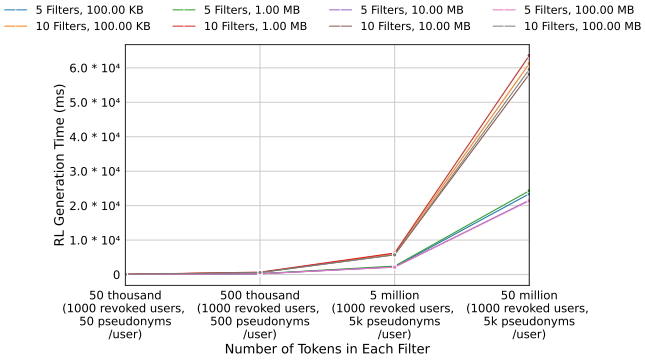
Both of our proposed techniques introduce additional time to the revocation list generation process. To evaluate the practicality of these techniques, we conducted a series of experiments aimed at assessing their impact on generation time. The goal was to determine whether the overhead introduced remains within acceptable limits—ensuring that the delay does not become prohibitively high or hinder the timely revocation of misbehaving clients. These tests help us understand the trade-offs involved and confirm the feasibility of implementing the proposed approaches in real-world scenarios.

5.5.1 HBFA. Figure 12 illustrates the results of a series of tests we conducted to assess the additional overhead involved in generating the revocation list using the Hierarchical Bloom Filter Arrays technique, as well as its impact on the latency required to generate a revocation with this approach.

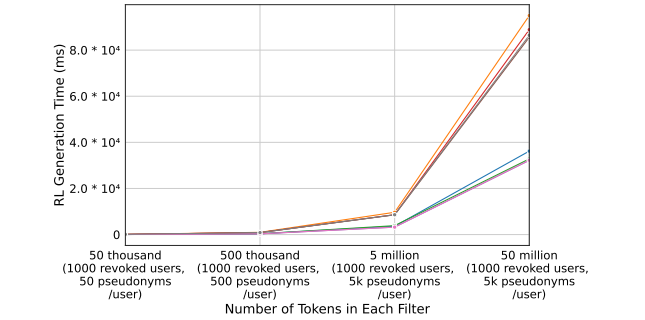
There are several factors that may impact the necessary additional overhead in generating a revocation list with multiple Bloom filters, instead of just one. We have conducted a series of tests to measure the additional latency this would take, varying parameters such as: the number of filters that our revocation list was composed of, the number of hash functions of each filter, the size of those filters and the number of tokens to be inserted in each filter.

By analyzing our results it is possible to notice that the number of hashes, the number of tokens to be inserted and the number of filters, have a linear impact in the additional overheard, doubling the latency when doubling the number of hashes or the number of tokens, due to the number of bits that are necessary to set to "1" also double. We can also conclude that the size of the Bloom filters do not have a significant impact on this metric, due to the nature of Bloom filters being highly efficient and constant with respect to their operations.

Also, we can conclude that, depending on the system scale, the additional overhead can be negligible, being only a few milliseconds



(a) 5 hash functions



(b) 10 hash functions

Fig. 12. HBFA's Overhead

for most cases until 5 million tokens, starting to impose a serious overhead when we hit the scale of 50 million tokens, reaching the magnitude of 60 seconds or more.

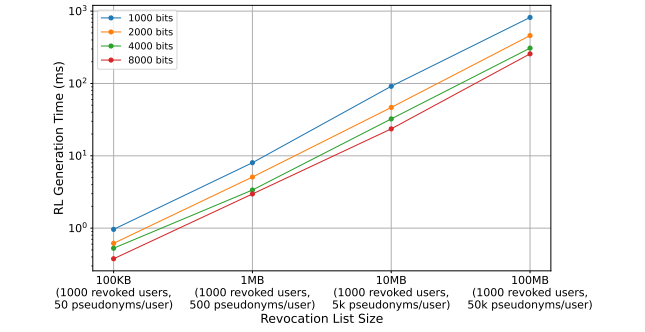


Fig. 13. Redactable Signature Overhead.

5.5.2 RS. To quantify the overhead introduced by our Redactable Signature technique, we conducted experiments measuring the time required to generate a signature over a revocation list, varying both the size of the Bloom filter and the size of the chunks into which it was divided.

As shown in Figure 13, the signature generation time increases linearly with the Bloom filter size, as expected. Additionally, using smaller chunks results in longer generation times due to the larger number of leaf nodes in the Merkle tree, which increases the number of required hash computations. In contrast, larger chunks yield smaller trees and thus lower computational overhead. The latency required to generate a redactable signature ranges from just 1 millisecond or less for Bloom filters of 100KB to hundreds of milliseconds for filters as large as 100MB.

This highlights a trade-off between redaction granularity and efficiency. As discussed before, (see Figures 8 and 9), larger chunks reduce signature generation time but require more data transfer during authentication. This trade-off becomes particularly relevant for Bloom filters exceeding 100MB. For smaller filters, generation time remains below one second across all chunk sizes, making the overhead negligible in practice.

6 Conclusions

In this work, we addressed the problem of efficiently ensuring revocation auditability. In previous work, this property was ensured by sending the client the complete revocation list so that they could verify their revocation status before authenticating in the system. This requirement could cause authentication to take up to several dozens of seconds, which is unacceptable for applications involving human interaction. We propose two techniques that allow for the transfer of less information, thus achieving an acceptable and practical authentication latency. Experimentally, we show that one of these techniques significantly reduces the authentication time (from 17 seconds to 2 milliseconds).

References

[1] Daniel Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of cryptographic engineering* 2, 2 (2012), 77–89.

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

[3] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer, 61–76.

[4] C. Correia. 2024. *Low-Latency Privacy-Preserving Access to Edge Storage*. Ph. D. Dissertation. Instituto Superior Tecnico, Universidade de Lisboa.

[5] Cláudio Correia, Miguel Correia, and Luís Rodrigues. 2023. Using Range-Revocable Pseudonyms to Provide Backward Unlinkability in the Edge. In *Proceedings of the ACM Conference on Computer and Communications Security. Copenhagen, Denmark*.

[6] Jason J Haas, Yih-Chun Hu, and Kenneth P Laberteaux. 2011. Efficient certificate revocation list organization and distribution. *IEEE Journal on Selected Areas in Communications* 29, 3 (2011), 595–604.

[7] Robert Johnson, David Molnar, Dawn Song, and David Wagner. 2002. Homomorphic signature schemes. In *Cryptographers' track at the RSA conference*. Springer, 244–262.

[8] Jiangtao Li, Ninghui Li, and Rui Xue. 2007. Universal accumulators with efficient nonmembership proofs. In *Applied Cryptography and Network Security: 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007. Proceedings 5*. Springer, 253–269.

[9] Patrick P Tsang, Man Ho Au, Apu Kapadia, and Sean W Smith. 2008. PEREA: Towards practical TTP-free revocation in anonymous authentication. In *Proceedings of the 15th ACM conference on Computer and communications security*. 333–344.

[10] Patrick P Tsang, Apu Kapadia, Cory Cornelius, and Sean W Smith. 2009. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Transactions on Dependable and Secure Computing* 8, 2 (2009), 256–269.

[11] Eric Verheul, Christopher Hicks, and Flavio D Garcia. 2019. Ifal: Issue first activate later certificates for v2x. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 279–293.

[12] Yifeng Zhu, Hong Jiang, and J. Wang. 2004. Hierarchical Bloom filter Arrays (HBA): a novel, scalable metadata management system for large cluster-based storage. In *Proceedings of the IEEE International Conference on Cluster Computing*. San Diego (CA), USA, 165–174.