RFID Batch Authentication—A Usable Scheme Providing Anonymity

PRZEMYSŁAW BŁAŚKIEWICZ¹, BARTOSZ DRZAZGA¹, ŁUKASZ KRZYWIECKI¹, DAMIAN STYGAR², AND PIOTR SYGA³

¹Department of Fundamentals of Computer Science, Wrocław University of Science and Technology, 50-370 Wrocław, Poland
²Department of Telecommunications and ICT, Wrocław University of Science and Technology, 50-370 Wrocław, Poland
³Department of Artificial Intelligence, Wrocław University of Science and Technology, 50-370 Wrocław, Poland

Corresponding author: Łukasz Krzywiecki (lukasz.krzywiecki@pwr.edu.pl)

This work was supported in part by the Fundamental Research Fund of the Wrocław University of Science and Technology under Grant 821104160.

ABSTRACT Businesses are slowly replacing Universal Product Code (barcodes used for tracking items) with radio-frequency identification (RFID). Unlike a barcode, RFID tag does not need to be within the line of sight of the reader, and can be interrogated from up to hundreds of meters from the reader in the case of active tag. Some readers are designed to read multiple RFID tags concurrently, which is useful in, e.g., warehouses. We address the problem of efficient methods for simultaneous authentication of multiple RFID tags – batch tag authentication. We provide a general design to build a provable secure batch tag authentication protocol from a single-tag authentication scheme using some aggregation function. Our main goal is to build a computationally-efficient batch tag authentication method that provides a reasonable level of anonymity in a model with an untrusted RFID reader. We provide an efficiency analysis of a protocol where the Bloom filter is used as the aggregation function together with a brief comparison with alternative data structures.

INDEX TERMS Batch authentication, bloom filter, cuckoo filter, RFID.

I. INTRODUCTION

Over the last two decades the Radio Frequency IDentification has gained considerable attention both in scientific as well as business circles. By its name, this technology allows for automated identification by reading special identifiers (tags) using radio waves. Thanks to this, the technology provides a considerable advantage over other methods like barcodes in that, it allows out-of-sight (through a radio wave permissive materials, such as paper, glass, etc.) and long-distance (up to a few meters) readouts.

On top of the RFID technology, the Electronic Product Code (EPC) [7] has been designed to replace the UPC (Universal Product Code) to facilitate business operation for merchant goods. Notably, the RFID concept also found other uses, such as tracking animals, personnel activity logging, and many more. In accord with the requirements, several standards have been proposed, among which the EPC-compliant RFID tags found their prominent role.

The associate editor coordinating the review of this manuscript and approving it for publication was Renato Ferrero.

The EPC-Gen2 RFID tags have several features that make them more flexible than just mere labels for items. While they are capable of storing permanent identifiers (that of the producer, as well as the unique product identifier – see [7]) in ROM, they also have a certain amount of read-write memory available to the user, for storing arbitrary information. Access to that memory can be password protected, effectively allowing only trusted parties to interact with the tag’s memory. The tags can also perform basic operations, among which random number generation (RNG function) and 16-bit CRC error correction code calculation are mandated by the EPC-Gen2 specification.

Generally, the tags are considered computationally limited devices. They can perform such operations as bitwise-XOR and summation, but cryptographic capabilities of tags remain very scarce and are under heavy development. There are two main factors contributing to that case: firstly, the available on-chip area is limited to keep the per-unit price as small as possible (note that for tracking and identification of goods the tags are essentially of single-use). The other factor is that the energy and clock for computation are harvested from...
the reader’s transmission. Consequently, complex operations would require higher reading powers applied over longer periods, or drastically limiting the reader-tag distance, both of which may not be a desirable effect.\footnote{Described here are \textit{passive} tags, which carry no battery and are entirely reliant upon the reader’s communication. In contrast, the \textit{semi-passive} or \textit{active} tags have their own sources of energy (batteries) to support calculation (former) and calculation and transmission (latter). Such tags are however beyond the scope of this paper.}

The remaining two parts of the RFID identification system are readers and back-end servers. Readers interrogate tags and extract information stored in them. They can take the form of a hand-held device or a matrix of antennas arranged so that a sensing area is formed. Readers can interrogate tags using the following sequence of commands:

- **select** selecting a subset of tags in the reading range for further processing;
- **inventory** identification of particular tags, selected in the previous step; this is where the tags present their EPC codes;
- **access** reading/writing from/to particular tag’s memory; here access to password-protected memory is possible by providing an adequate key.

Servers maintain databases with entries for each tag and supply readers with the information required for processing tags. This allows for tags to be processed in many distinct places, while the information is kept uniquely in one place.

It is also typically assumed that readers and servers can perform computationally extensive operations so that, for example, it is possible to authenticate readers to database servers to assure that passwords for accessing tags are issued to legitimate parties. In contrast, tags are considered the weakest link in the chain, with their limited computation and memory capabilities, yet are on the front line against attacks and malicious use. In that setting, some problems arise, and similarly, vast body of solutions have been provided by researchers.

**A. SINGLE AND BATCH OF TAGS ARCHITECTURES**

In this paper, we address only one of such problems. To protect against counterfeit, suppose a manufacturer labels its goods with a specific set of RFID tags, which identifiers (EPCs) are stored in the database and allow the products to be verified for originality.

1) **BASIC TAG AUTHENTICATION SCENARIO**

Since a tag with the same EPC can be easily produced, the manufacturer must use other means to safeguard their set of tags. They will store a secret value (key) in each tag’s password-protected memory. When challenged with a nonce by the reader, the tag will operate on its key and the nonce, producing a \textit{pseudonym}, which it will then send back along with its EPC. Next, the challenge nonce, the pseudonym, and the EPC can be forwarded to the producer’s database, where the key corresponding to the given EPC is retrieved, and an attempt to recalculate the pseudonym is made. If the results match, it can be assumed that the tag is legitimate.

Given the limited computation capabilities of the tag, such a scheme is feasible when, e.g., tag-optimized one-way hash functions are used to process the key and the nonce. On the other hand, when there is more than one tag (a batch) to be verified, then for each tag there is a communication between the reader and the back-end database. This can result in a prohibitively long time for authentication of the entire batch. Therefore, some aggregation function can be used on all tags’ responses, and only its result is forwarded to the server. The approach that takes advantage of an aggregation function allows to still use some single-tag authentication protocol (presumably having very low computational requirements) and limits operations on the backend server side. In the case of the aggregated response, the server needs to perform only one exhaustive database search, while in the case of separate single-tag authentication the server would need to perform a database search for each of the tags. The tradeoff is that the reader needs to perform additional calculations to aggregate the responses. However, the reader is not as computationally constrained as the tags and aggregation allows to greatly speed up operations performed by the server.

2) **BATCH-AGGREGATION AUTHENTICATION PROTOCOL**

We present a protocol for such a system with the following functional properties:

1) A \textit{batch} of tags is interrogated by the reader where each tag presents its pseudonym.

2) Pseudonyms are \textit{aggregated} by the reader and forwarded to the back-end server, where it can be decided:
   a) whether only legitimate tags have contributed their pseudonyms, and
   b) which tags have done it.

We derive our batch authentication scheme from a single-tag authentication protocol and show that, assuming this basic scheme is secure, our extension inherits the security. We support our claims with adequate calculations and provide simulation results.

a) **PROBLEM STATEMENT AND OUR CONTRIBUTION**

In a batch authentication protocol the following, seemingly mutually exclusive problems arise:

1) \textit{Anonymity}. The responses should guarantee a fair level of privacy assumed in the anonymity model. For example, no recorded transcript of messages (passive adversary model) should allow linking multiple protocol executions to a particular tag, or a particular batch of tags.

2) \textit{Self-containment}. On the other hand, the group identification process should allow the identification of every tag in the batch. No false or dummy tags should pass the verification process. In the minimalistic setup, the operations in the protocol require only one shared input secret key. The resulting aggregation of responses
should precisely define the subset of tags included in the batch.

3) **Effectiveness.** The server needs to verify tags' responses against each entry in its database sequentially to discover counterfeited tags. The amount of information exchanged between the reader and the server grows with the size of the batch.

This paper is an extended version of [2], whose main contributions are:

- We present downsides of a solution for the batch authentication problem as given in [4] and offer our own that makes the authentication and identification feasible in the considered scenario.
- We introduce a protocol for batch authentication (IBS), which leverages a single-tag authentication protocol (IS) with the following assumptions:
  - server database consists of \( z \) records;
  - size of a batch is \( n \ll z \);
  - tag identification is possible and retains anonymity to the external adversary;
  - reader can receive concurrent responses from many tags.
- We discuss the security of the general construction of IBS and prove it is secure and anonymous in our model.
- We identify the main problems for this scenario – the length of the batch message from the reader to the server, and the complexity on the server side. If the message is "compressed" too much and there is no adequate definition of the set of tags in the batch, then to verify positively under our assumptions the server has to check \( \binom{z}{n} \) potential subsets. Assuming even small numbers, e.g., \( n = 30, z = 1000 \) this number exceeds \( 2.4 * 10^{57} \), rendering the protocol unusable.
- As a solution, we present a batch authentication protocol based on Bloom Filters that reduces computation overhead when compared to [4] and greatly reduces the communication overhead in comparison to a naive "listing of the batch". We provide formal analysis, showing the upper bounds on the number of server operations required to identify the batch as well as the upper bound on the message length.

This manuscript extends the conference paper in the following aspects:

- We refine the security models for regular and batch identification schemes, taking into account the passive and active adversaries during the query and attack stages of security experiments.
- We provide more thorough security proofs, which were only briefly sketched in the conference paper, due to the space constraints. In our security proofs, we reduce the security of a batch tag authentication protocol to the security of a single tag authentication scheme.
- Additionally to the base Bloom filter technique, we analyzed double hashing and cuckoo filters potential usage in the compression function between the reader and the server. We discuss the communication and computational complexity on the server – correlated to false positive ratio of the filters. Cuckoo filter is a data structure that supports approximate set membership tests and can replace Bloom filters. Cuckoo filter has been applied in, e.g., vehicular ad-hoc networks (VANETs) for more efficient authentication protocols [27], [28], Software Defined Networking (SDN) use-cases like network caching, multicast, and firewalls [12], and Named Data Networking (NDN) [20].
- Finally we implement our propositions based on: regular Bloom filters, double hashing, and cuckoo filters. We discuss the obtained results, taking into account the compression advantages, as well as the computational complexity on the server-side, and provide comments on potential usage for batch identification scenarios.

**B. RELATED WORK**

As mentioned above, there is a vast body of literature concerning systems of RFID tags, their security, and efficient algorithms. [1] is a survey of authentication protocols for RFID tags. The authors of [31] propose a protocol based on a group signature that allows authentication of pre-determined batches of tags. We note that we focus on a flexible approach that allows authentication of any subset of tags in the database. Several seminal ideas for tags’ security and privacy can be found in [30]. Among others, the authors propose a method called Hash Lock. It consists of applying a hash function to the tag’s ID and a random value to generate a temporary identifier, which is later presented to the database to discover the original ID by exhaustive search over all possible identifiers. The authors of [16] provide a good overview of research aimed at alleviating the obvious problem of such an approach, i.e., linear in the number of tags complexity of the search for the original ID of the tag. They also present a protocol that organizes the keys into a tree structure allowing more effective search operations and, at the cost of increased storage requirement on the tag’s side, provides logarithmic search time for the database.

Another approach to counterfeit discovery in a batch of tags is presented by the authors of [33] in their SEBA (Single Echo Batch Authentication) protocol and in a more recent FISH protocol [18]. In these proposals, the tags are presented with a nonce which they use along with their secret keys to determine a time-slot for responding in the framed slotted ALOHA protocol. The no-transmission, single-transmission, and collision slots sequence is then analyzed to determine any outlying tag responses; the assumption being that a tag with no valid key (i.e., one that is not stored in the database) will respond in a slot that otherwise would be a no-transmission slot, or cause a collision in an otherwise single-transmission slot. The authors admit that analyzing all possible transmission sequences for a large pool of possible tags is overly complex and provide a simplification of the protocol allowing faster rejection of a batch with a counterfeited tag.

---

\(^2\)The full version of the paper was presented in ISPEC 2016 [2]
In the two papers [9], [10], the authors also utilize the idea of determining outlying (counterfeit) tags by analyzing the sequence of single-transmission, no-transmission, and collision slots in the slotted ALOHA protocol in a scenario with a single (former) and multiple readers (latter). The difference from work in [33] being that here a statistical approach is employed and estimates for the cardinalities of the sets of legitimate and counterfeit tags are provided. Similar approach can be found in [25], where slotted-ALOHA scheme is used for authentication of the batch. However, in this solution the authors introduce the concept of groups of tags and corresponding group keys. When challenged with a nonce by the reader, the tags reply with a hash of that nonce concatenated with their group key. Since the number of group keys is much smaller than the number of possible tags, the reader can exhaustively search for that key and, consequently, learn which groups of tags have responded to the query. This in turn limits the number of tags that have to be authenticated in the subsequent stage using slotted-ALOHA technique.

In [26] authors present an update to YA-TRAP [29] authentication scheme to facilitate its use for batch of tags. In their proposal, the tag sends an authentication hash derived from two values: hashed challenge submitted by the reader and a pseudo-random number generated from a subsequent run of its internal PRNG. That number is sent along with the authentication hash to the reader, where hashes from all tags in a batch are XOR’ed and all random numbers concatenated. This data is next sent to the server, where an attempt is made to redo the hashing operation based on the supplied randomness. The application of this scheme to the batch authentication seems to be only in limiting transmission overhead.

Notably, since the publication of our original paper [2] a similar approach to ours has been presented in [17], also utilising Bloom filters. However, this work differs from our approach in the assumption that the database of legitimate tags is small enough to fit in a reasonably short filter without much loss of information. In this paper we also expand our paper with other than Bloom filter methods for dealing with large sets of tags.

The authors of [13] developed a batch authentication protocol with healthcare industry in mind. Their proposal uses Dynamic Framed Slotted ALOHA (DFSA). Aforementioned SEBA protocol has a certain probability of authentication error, which renders that protocol unusable in scenarios requiring high accuracy like medical equipment communication. The protocol from [13] can find every illegal tag while realizing batch authentication. Its main contributions are that the back-end server does not store keys for all tags in the system, and the computing power requirements of the tags are very low because the protocol uses the properties of homogeneous linear equations.

In a recent paper [32] authors present a deterministic protocol for authentication of batches of tags, when the modulation technique for backscatter transmission allows for collision detection by the reader. It is assumed that a preset group of tags in a batch will respond to the reader’s interrogation in a specific form, yielding a pre-determined collision pattern in the response. Since each tag in a group contributes to collisions at positions determined by a secret pre-shared with the reader, the latter can authenticate the entire group by verifying occurrences of collisions in the response.

Apart from purely algorithmic approaches to the problem of batch authentication, notable research has been made to exploit intrinsic, technical properties of tag’s radio emissions (fingerprinting). In [34] authors introduce the idea of phase fingerprint and present a three-stage scheme where the fingerprints are created, then a pre-determined set of tags is attached to the labelled item and finally the collective fingerprint along with the geometry of tags’ placement is investigated to assess the product’s genuineness. Recently proposed B-AUT protocol [35] takes this concept further by introducing a pre-authentication phase where a clustering algorithm is used to sub-divide tags in the batch into more manageable subsets upon which a precise fingerprint matching is then executed. Moreover, the B-AUT is shown to have the capability of indicating the position of fraudulent tags in the batch.

While this paper improves mainly on the proposal from [4], which is described in detail in the next section, we note that most of the above papers deal with an expected, fixed a-priori set of tags in the authenticated batch. While valid in some scenarios, such as supply-chain monitoring, this approach cannot be applied when each batch may contain an arbitrary subset of tags, as may be the case, e.g., in automated cashier machines which scan shopping baskets. And in contrast to [32], [35] we make no underlying assumptions about technical aspects of tags’ communications.

II. PREVIOUS BATCH RFID IDENTIFICATION

In this section we will underline the issues related to the problem of batch authentication of tags using the protocol from [4] as an example. Next, we will provide solutions to these problems, thus formulating our algorithm for batch authentication. The algorithm in [4] considers an RFID system consisting of a Reader (R), a database and a fixed set of n tags T_1, ..., T_n, each sharing a unique key k_i with R and DB. Every tag is preloaded with a value w by the batch creator (a “batch identifier”), however this value is different in each tag in the batch. The batch identification session of this proposal is shown in Fig. 1, which is Figure 2 in the original paper.

The authentication procedure begins when R challenges tags with a nonce c. The tags use their batch identifier w, challenge c and their secret keys k_i to calculate value R[i] for i-th tag, where i ∈ {1, ..., n}. Arguably, this step of the protocol is not clearly described, because, e.g., the reason for updating ∆_0, ∆_1 is not clear at all. Nevertheless, the subsequent flow is clear: the reader collects all R[i] and generates M as given in Fig. 1, which is sent to the server. Here, the server somehow calculates its value of R (the authors do not specify the means by which this is achieved). It can only be assumed that the server performs the same operations as each tag in the batch, using c specified by R (the authors do not specify if c is
FIGURE 1. Batch authentication protocol from [4] (Fig. 2 therein).

- Reader → tags : The reader sends a challenge message \( c \) to all tag to initiate a session.
- Tags \( t_i \) → reader : Let \( V[0] \leftarrow \text{CRC}(w) \oplus S \), and \( \Delta_0 \leftarrow 3 \cdot \text{CRC}(w) \Delta_1 \leftarrow 2 \cdot \text{CRC}(w) \).  
  After receiving the challenge message, for each tag \( t_i \in T \) parse \( k_i \) with \( c \) runs its internal PRNG \( a_i = G_i(c \oplus k_i) \). 
  \( V[i] \leftarrow G(a_i \oplus \Delta_0) \oplus V[i-1] \) 
  \( R[i] \leftarrow G(V[i]) \oplus \Delta_1 \) 
  \( \Delta_0 \leftarrow 2 \Delta_0, \Delta_1 \leftarrow 2 \Delta_1 \).
  The tag \( t_i \) sends \( R[i] \) to the reader.
- Reader → back-end server: The reader gathers all the message from the tags and computes \( M = E_k(E_b(\Sigma \oplus 2^{d-1}3^2S) \oplus R) \oplus 2^{d-1}7S \) and sends \( M \) to the back-end server.
  Here, \( R = R[1] \oplus R[2] \oplus \cdots \oplus R[n] \) and \( \Sigma \overset{\text{def}}{=} k_1 \oplus k_2 \oplus \cdots \oplus k_n \).
- Server processing: The back-end server receives \( M \) from the reader who gathers all the response from tags. The server performs exactly the same operations on its side, since the operation of un-masking is just simply XORing the masked value with the mask.

Therefore, the only way the verification can fail is when the values \( R, \Sigma \) used by the reader are different from that used by the server. Since the latter is pre-defined by the batch content, the only error can arise when \( R \) is different. After the investigation of the scheme from [4] we identified the following problems:

- Some of the operations made by \( R \) and the server seem irrelevant concerning the usability of the protocol. It is only \( R \) that carries any information about the batch – there is little point in transferring other data to the server.
- \( R \)’s knowledge of keys \( k_i \) for each tag implies a strong assumption about honesty: should some tags be missing from the batch, the message \( M \) can still be tailored by the reader to show as if these tags were present.
- Letting the reader know all \( k_i \)’s renders this scheme inapplicable in situations where any reader should be able to identify the legitimacy of a batch with the help of the database: the reader should be considered merely a proxy between the batch and the database, instead of becoming an active side.
- It is not clearly described in the paper how the value \( R \) is obtained on the server-side. It may be assumed that the server calculates it based on the challenge \( c \) (provided it is sent by \( R \)). If so, then, essentially, the server performs exactly the kind of processing that would be done if the batch was inspected on a tag-by-tag basis: it follows the calculation of tag’s and verifies if it matches any of the received \( R[i] \). This makes the “batching” process redundant and unnecessary overhead.
- Determining if all specified tags are in the batch is one problem, but a slightly more adequate problem is to determine if all tags that can be scanned belong to that batch. This might be particularly the case when, due to changes in relative positions of the tags in the batch and the reader, some of the tags become non-responsive (shaded). Then, the proposed solution will fail, since some of \( R[i] \)’s won’t be registered making the value \( R \) different from what will be used by the server.
- The scheme can be used only for a predetermined batch of tags, e.g., to check if no tags have been lost during transportation. This limits the applicability of the protocol.

In what follows, we will give solutions to solve these problems and provide adequate proofs for our proposals. Namely, we will show how a batch can be authenticated without the reader knowing tags’ secret keys. Also, we will show how the same amount of computation made by the server can be utilized to obtain a far better result: that of verification of genuineness and identification at the same time of an unknown subset of tags from the batch.

III. EXTENDED BATCH IDENTIFICATION

In this chapter, we present our protocol and provide strict proofs of its security. But before that, let us introduce a more formal description of the model we are dealing with.
A. NAMING CONVENTIONS

Let our system consist of a reader (R), a server (S), and a set of all possible tags (Ω), and z = |Ω|. A single tag is a device profiled by a unique key ki, i.e., such a key is “inserted” into the tag at the beginning of its lifespan, and since then the key is used to identify that tag. By ti we denote a tag ti(ki) with the key ki inside. Since z = |Ω| we assume that keys indexes i ∈ {0, . . . , z − 1}. Moreover, let the reader and the server share a key k that will be used for encrypting the communication between them using symmetric encryption scheme (E, D), where E and D denote encryption and decryption, respectively. Finally, let π be the authentication protocol run between R, S, and some subset of Ω tags (a batch) of size n. Let y1, . . . , yn ← Y denote each yi is sampled uniformly at random from the set Y.

Here, a symmetric-key algorithm cryptosystem is used for securing the communication between the reader and the server. It does not influence the security of the tags identification process, and is used only for secrecy of the reader-server communication. However, for the self-containment of the paper, we recall the Indistinguishability under Chosen Plaintext Attack (IND-CPA) model for symmetric-key algorithm cryptosystems from [14], which is equivalent to the older semantic security model of [11]. Namely, an adversary should not compute any information about a plaintext from its ciphertext, i.e., an adversary, producing two plaintexts of equal length and obtaining just one respective ciphertexts, cannot determine which plaintext was encrypted. Indistinguishability under Chosen Plaintext Attack (IND-CPA) is commonly defined by the following experiment:

**Definition 1**: A symmetric-key cryptosystem is a tuple (ParGen, KeyGen, E, D), where:

- par ← ParGen(λ): takes the security parameter λ and outputs parameters par = (K, M, E), where K is a keys space, M is a messages space, E is a ciphertexts space.
- k ← KeyGen(par): is a key generation algorithm, which inputs par and outputs a symmetric key k ∈ K.
- e ← Ek(m): is an encryption algorithm that takes a symmetric key k, inputs m ∈ M, and outputs a ciphertext e ∈ E.
- m ← Dk(e): is a decryption algorithm that takes a symmetric key k, inputs e ∈ E, and outputs the corresponding plaintext m.

s.t. for any k ← KeyGen(par), and any message m ← M, if e ← Ek(m) then, m ← Dk(e).

**Definition 2** (Indistinguishability Under CPA): Given a symmetric-key cryptosystem (ParGen, KeyGen, E, D) we define the chosen plaintext indistinguishability experiment IND-CPA:

- Init: par ← ParGen(λ), k ← KeyGen(par).
- Adversary: Let the adversary A be a malicious algorithm initialized with the parameters par.
- Challenger: Let the challenger C be an algorithm initialized with the parameters par, and the secret encryption/decryption key k.

• **GuessGame**: It is the following protocol:
  - The adversary generates two messages of its choice: (m1, m2) ← A(par) and sends them to the challenger.
  - The challenger C(par, k, m1, m2) generates a random bit b ← {0, 1}, encrypts the message mb to the ciphertext e = Ek(mb), and sends e to A.
  - The adversary outputs its own bit

\[
\tilde{b} \leftarrow A(par, m_0, m_1, e)
\]

We define the advantage of the adversary A in the experiment as the probability that the A outputs the correct bit \(\tilde{b} = b\) indicating the encrypted message mb, i.e.:

\[
Adv(A, IND – CPA) = |Pr[\tilde{b} = b] – 1/2|.
\]

We say that the encryption scheme is IND – CPA secure if the advantage of the adversary A is negligible in parameters λ, i.e.:

\[
Adv(A) \leq \epsilon(λ).
\]

Now we begin to construct our abstracts, representing authentication protocols. We start with a definition of a single-tag identification scheme and introduce the concepts of impersonation and anonymity that we require that the single-tag identification scheme possesses. Next, we move on to definitions of a batch identification scheme as an extension to the single-tag scheme and consequently adapt impersonation and anonymity properties. Finally, we provide a general construction for the batch identification scheme and show that it is secure if the underlying single-tag identification is secure.

B. SINGLE TAG IS SCHEME

**Definition 3** (Tag Identification Scheme): An identification scheme IS is a system which consists of five algorithms (ParGen, KeyGen, t, R, S) and a protocol π:

- params ← ParGen(λ): inputs the security parameter λ, and outputs public parameters available to all users of the system, thus we omit them from the rest of the description.
- (ki, k) ← KeyGen(): outputs secret keys ki, k.
- t(ki): denotes the tag – an ITM which on input of the key kiti interacts with the reader R(ki).
- R(ki): denotes the reader – an ITM which on input of the key ki interacts with the tag t(ki) and the server S(ki, k) in protocol π.
- S(ki, k): denotes the server – an ITM which on input of the keys k, ki, interacts with the tag t(ki) and the reader R(ki) in protocol π.
- π(t, R, S): denotes the protocol between the tag, the reader, and the server.

There are two stages of the scheme:
We require that the scheme is complete: π → ParGen(1^k), devices are registered, e.g., on behalf of the tag t, the procedure k_i ← KeyGen(i) outputs the secret key that the tag shares with the server.

2) Operation: In this stage any tag, e.g., t_i, demonstrates its identity to the server by performing the protocol π(t(k_i), R(k), S(k, k_i)) related to the keys k, k_i. Finally the server outputs 1 for “accept” or 0 for “reject.” For simplicity we denote π(t(k_i), R(k), S(k, k_i)) → 1 if t_i was accepted by S in π.

We require that the scheme is complete, i.e., protocol π(t(k_i), R(k), S(k, k_i)) returns 1 for any pair (k_i, k) ← KeyGen(i).

A scheme is secure if it is impossible for any adversary algorithm A to be accepted by the server as t_i without the knowledge of the correct k_i. That is from the secure system we require that Pr[π(A(t_i), R(k), S(k, k_i)) → 1] is negligible. Typically, the security model distinguishes passive and active adversary modes of operation. The passive adversary in the query stage of the security experiment passively listens to protocol executions, trying to gain some knowledge about the system, and somehow learns the data which would allow him the successful impersonation later on. This mode includes the scenario where the adversary collects the transcripts of protocols via appropriate recording device. The active adversary in the query stage can chose adaptively challenges which are sent to the tag. In this strong scenario, for a number ℓ of repetitions, the adversary chooses the challenge which is to be sent by the reader. The adversary observes the protocol executions for these challenges, i.e., answers returned by the tag, and the corresponding messages sent from the reader to the server.

Definition 4 (Impersonation (I)): Let IS = (ParGen, KeyGen, t, R, S, π) is a tag identification scheme. We define a security experiment Exp^{K, ℓ}_IS as follows:

- Init stage: Let params ← ParGen(1^k), (k_i, k) ← KeyGen(i). Let A denote the adversary algorithm.
- Passive Query stage (PQ): A passively observes a polynomial number ℓ of executions of the protocol π(t(k_i), R(k), S(k, k_i)). Let v_ℓ^i = {T_1, . . . , T_ℓ_i} be the view A gains after the ℓ runs of π(t(k_i), R(k), S(k, k_i)), where T_j is the transcript of j-th execution.
- Active Query stage (AQ): A actively chooses challenges c for the tag in a polynomial number ℓ of runs of protocol executions. Let T_j = {c_j ← A()}, π(t(k_i), R(c_j, k), S(k, k_j)) denotes the knowledge the adversary gains in the j-th execution. Let v_ℓ^i = {T_1, . . . , T_ℓ_i} be the view A gains after the ℓ runs of the protocol.
- Impersonation stage: A runs the protocol

\[ π(A(v_ℓ^i), R(k), S(k, k_j)) \]

with the reader and the server.

We define the advantage of A in Exp^{K, ℓ}_IS as the probability of acceptance in the last stage:

\[ \text{Adv}(A, \text{Exp}_{\text{IS}}^{K, ℓ}) = \text{Pr}[π(A(v_ℓ^i), R(k), S(k, k_j)) → 1]. \] (3)

We say that the scheme IS is secure if Adv(A, Exp_{IS}^{K, ℓ}) ≤ ε(λ).

Note that in the active model, the adversary does not know the key k shared between the reader and the server, however it can choose the challenges, which somehow can turn out to be advantageous.

Now, let us define the anonymity property of a tag. Consider two tags t(k_0), t(k_1) with two distinct keys k_0, k_1. The scheme is anonymous if the adversary cannot distinguish if in the challenge stage, the protocol is executed either with the tag t(k_0), or with the tag t(k_1). Similarly to previous definition we consider passive and active modes of operation in the query stage. Additionally, we also consider two modes of operations for the challenge stage. In passive challenge mode, the adversary passively observes the protocol. In active challenge mode, the adversary chooses the challenge for the protocol execution in the challenge stage.

Definition 5 (Anonymity (A)): Let IS = (ParGen, KeyGen, t, R, S, π) is a tag identification scheme. We define security experiment Exp IS^{A,K, ℓ}:

- Init stage: Let params ← ParGen(1^k), (k_0, k_1, k) ← KeyGen(i). Let A denote the adversary algorithm.
- Passive Query stage (PQ): A observes a polynomial number ℓ:

  - of executions of the protocol π(t(k_0), R(k), S(k, k_0)) for the tag t(k_0). Let v_ℓ^0 = {T_{0,1}, . . . , T_{0,ℓ}} be the view A gains after the ℓ runs of π(t(k_0), R(k), S(k, k_0)), where T_{0,j} is the transcript of i-th execution.
  - of executions of the protocol π(t(k_1), R(k), S(k, k_1)) for the tag t(k_1). Let v_ℓ^1 = {T_{1,1}, . . . , T_{1,ℓ}} be the view A gains after the ℓ runs of π(t(k_1), R(k), S(k, k_1)), where T_{1,i} is the transcript of i-th execution.

- Active Query stage (AQ): A actively challenges the tags in a polynomial number ℓ executions:

  - of the protocol with t(k_0): Let T_{j,0} = {c_j ← A()}, π(t(k_0), R(c_j, k), S(k, k_0)) denotes the knowledge the adversary gains in the j-th execution. Let v_ℓ^0 = {T_{1,0}, . . . , T_{0,ℓ}} be the view A gains after the ℓ runs of the protocol.
  - of the protocol with t(k_1): Let T_{j,1} = {c_j ← A()}, π(t(k_1), R(c_j, k), S(k, k_1)) denotes the knowledge the adversary gains in the j-th execution. Let v_ℓ^1 = {T_{1,1}, . . . , T_{1,ℓ}} be the view A gains after the ℓ runs of the protocol.

- Passive Challenge stage (PC): The challenger C draws a bit at random b ← {0, 1}, and runs the
protocol \( \pi(t(k_b), R(k), S(k, k_b)) \) in front of the adversary. Let \( T_0 \) denote the transcript. Then, \( A \) outputs a bit \( A(\nu_0, \nu_1, T_b) \to \hat{b} \).

**Active Challenge stage (AC):** The challenger \( C \) draws a bit at random \( b \leftarrow \{0, 1\} \), and the adversary chooses the challenge \( c \leftarrow A() \). The protocol \( \pi(t(k_b), R(c, k), S(k, k_b)) \) is run in front of the adversary. Let \( T_0 \) denote the transcript. \( A \) outputs a bit \( A(\nu_0, \nu_1, T_b) \to \hat{b} \).

We define the advantage of \( A \) in \( \text{Exp}^{A, \lambda, t}_{\text{IS}} \) in a combination of modes (\( \text{FQPC}, \text{PQAC}, \text{AQPC}, \text{AQAC} \)) as probability of outputting the same bit as the challenger in the appropriate challenge stage:

\[
\text{Adv}(A, \text{Exp}^{A, \lambda, t}_{\text{IS}}) = | \Pr[b = \hat{b}] - 1/2 |. \tag{4}
\]

The identification scheme IS is anonymous if

\[
\text{Adv}(A, \text{Exp}^{A, \lambda, t}_{\text{IS}})) \leq \epsilon_A(\lambda).
\]

That is, the anonymity property is preserved when, having listened to \( \ell \) sessions of authenticating two distinct tags \( t_0 \) and \( t_1 \), the adversary still cannot tell which of the tags has been interrogated during \( \ell + 1 \) session with probability greater than 0.5 (a fair guess).

## C. BATCH TAG IBS SCHEME

Now let us extend these definitions to a system for batch identification of tags \( T \subset \Omega \).

**Definition 6 (Batch Tag Identification Scheme):** A batch tag identification scheme IBS is a system which consists of five algorithms \( \text{ParGen}, \text{KeyGen}, T, R, S \) and a protocol \( \pi \):

- \( \text{params} \leftarrow \text{ParGen}(1^\lambda) \): inputs the security parameter \( \lambda \), and outputs public parameters available to all users of the system, thus we omit them from the rest of description.
- \( (k_i, k) \leftarrow \text{KeyGen}() \): outputs the secret keys \( k_i, k \).
- \( T(K) \): denotes the set of tags \( T = \{t(k_1), \ldots, t(k_n)\} \) – a set of ITMs which on input of the key \( k_i \) each, interacts concurrently with the reader \( R(k) \) and the server \( S(k, k) \) in protocol \( \pi \), s.t. \( K = \{k_1, \ldots, k_n\} \) denotes the set of keys for each tag in \( T \).
- \( R(k) \): denotes the reader – an ITM which on input of the key \( k \), interacts with the tags \( T(K) \) concurrently and the server \( S(k, K) \) in protocol \( \pi \).
- \( S(k, K) \): denotes the server – an ITM which on input of the keys \( K \), \( k \), interacts with the tags \( T(K) \) and the reader \( R(k) \) in protocol \( \pi \).
- \( \pi(T, R, S, \lambda) \): denotes the protocol between the tags, the reader, and the server.

We distinguish two stages of the scheme:

- **Initialization:** The parameters are generated: \( \text{params} \leftarrow \text{ParGen}(1^\lambda) \), devices are registered, e.g., on behalf of the tag \( t_i \) the procedure \( k_i \leftarrow \text{KeyGen}(\cdot) \) outputs the secret key that the tag shares with the server.

- **Operation:** In this stage each tag \( t_i \in T \) demonstrates its identity to the server by performing the protocol \( \pi(T(K), R(k), S(k, K)) \) related to the keys \( k, k_i \). The reader sends the challenge \( c \) to all tags in \( T \) and subsequently concurrently reads all \( r_i \) from all tags in \( T \). Then it forms a short batch message to the server: Finally, the server outputs 1 for “accept” or 0 for “reject.” For simplicity we denote \( \pi(T(K), R(k), S(k, K)) \to 1 \) if \( T \) was accepted by \( S \).

We require that the scheme is complete, i.e., for any pair \((k, k') \leftarrow \text{KeyGen}(\cdot) \) the protocol execution \( \pi(T(K), R(k), S(k, K)) \) returns 1.

Security definitions of the batch scheme follow those of the regular scheme. Note that the major difference between single-tag and batch identification schemes is that in the latter case it is the batch \( T \) being identified. However, the attacker will be successful if he manages to impersonate only a single tag or break its anonymity. Let us denote \( \bar{K} = K \setminus \{k_i\} \), for any \( k_i \in K \). Then we require that \( \Pr[\pi(\bar{A}(K), R(k), S(k, K)) \to 1] \) is negligible. Consequently, the definitions follow:

**Definition 7 (Batch Impersonation (BI)):** Let \( \text{IBS} = (\text{ParGen}, \text{KeyGen}, T, R, S, \pi) \) be a batch tag identification scheme. We define security experiment \( \text{Exp}^{\text{BI}, A}_{\lambda, t, \text{IBS}} \):

- **Init stage:** Let \( \text{params} \leftarrow \text{ParGen}(1^t), (K, k) \leftarrow \text{KeyGen}(\cdot), \) where \( K = \{k_i\}^n_t \) is a set of keys of individual tags. Let \( A \), denote the adversary algorithm.
- **Passive Query stage (PQ):** \( A \) passively observes a polynomial number \( \ell \) of executions of the protocol \( \pi(T(K), R(k), S(k, K)) \). Let the view \( \nu^t = \{T_1, \ldots, T_t\} \) denotes the knowledge \( A \) gains after the \( \ell \) runs of \( \pi(T(K), R(k), S(k, K)) \), where \( T_i \) is the transcript of \( i \)-th execution.
- **Active Query stage (AQ):** \( A \) actively chooses challenges \( c_j \) for tags in a polynomial number \( \ell \) of protocol executions. Let \( T_j = \{c_j \leftarrow A(), \pi(T(K), R(c_j, k), S(k, K))\} \) denotes the knowledge the adversary gains in the \( j \)-th execution. Let \( \nu^t = \{T_1, \ldots, T_t\} \) be the view \( A \) gains after the \( \ell \) runs of the protocol.
- **Impersonation stage:** \( A \) chooses the tag \( t(k_i) \in T \), and is given \( \bar{K} = K \setminus \{k_i\} \), where \( k_i \) is of the chosen tag. The following protocol is executed: \( \pi(A(\bar{K}, \nu^t), R(k), S(k, K)) \) with the adversary, the reader and the server.

We define the advantage of \( A \) in the experiment \( \text{Exp}^{\text{BI}, A}_{\lambda, t, \text{IBS}} \) as probability of acceptance in the last stage:

\[
\text{Adv}(A, \text{Exp}^{\text{BI}, A}_{\lambda, t, \text{IBS}}) = \Pr[\pi(A(\bar{K}, \nu^t), R(k), S(k, K)) \to 1]. \tag{5}
\]

We say that the scheme \( \text{IBS} \) is secure if \( \text{Adv}(A, \text{Exp}^{\text{BI}, A}_{\lambda, t, \text{IBS}}) \leq \epsilon_1(\lambda) \).

**Definition 8 (Anonymity (A)):** Let \( \text{IBS} = (\text{ParGen}, \text{KeyGen}, T, R, S, \pi) \) is a batch tag identification scheme. We define security experiment \( \text{Exp}^{A}_{\lambda, t, \text{IBS}} \):

- **Init stage:** Let \( \text{params} \leftarrow \text{ParGen}(1^t), (k_0, k_1, \ldots, k_{t-1}, k) \leftarrow \text{KeyGen}(\cdot) \). Let \( A \) denote the adversary
We define the advantage of Definitions 4 and 5. Moreover, assume that (\textit{A QAC})

\begin{itemize}
    \item \textbf{Passive Query stage (PQ)}: A passively observes a polynomial number \(\ell\) of executions:
        \begin{itemize}
            \item of the protocol \(\pi(T(K_0), R(k), S(k, K_0))\). Let \(V_0 = \{T_{0,1}, \ldots, T_{0,\ell}\}\) be the view \(A\) gains after the \(\ell\) runs of \(\pi(T(K_0), R(k), S(k, K_0))\), where \(T_{0,i}\) is the transcript of \(i\)-th execution.
            \item of the protocol \(\pi(T(K_1), R(k), S(k, K_1))\). Let \(V_1 = \{T_{1,1}, \ldots, T_{1,\ell}\}\) be the view \(A\) gains after the \(\ell\) runs of \(\pi(T(K_1), R(k), S(k, K_1))\), where \(T_{1,i}\) is the transcript of \(i\)-th execution.
        \end{itemize}
    \item \textbf{Active Query stage (A Q)}: A executes a polynomial number \(\ell\):
        \begin{itemize}
            \item of the protocol with \(T(K_0)\): Let \(T_{0,j} = \{c_j \leftarrow A\}, \pi(T(K_0), R(c_j, k), S(k, K_0))\) denotes the knowledge \(A\) gains in the \(j\)-th execution. Let \(V_0 = \{T_{0,1}, \ldots, T_{0,\ell}\}\) be the view \(A\) gains after the \(\ell\) executions.
            \item of the protocol with \(T(K_1)\): Let \(T_{1,j} = \{c_j \leftarrow A\}, \pi(T(K_1), R(c_j, k), S(k, K_1))\) denotes the knowledge \(A\) gains in the \(j\)-th execution. Let \(V_1 = \{T_{1,1}, \ldots, T_{1,\ell}\}\) be the view \(A\) gains after the \(\ell\) executions.
        \end{itemize}
    \item \textbf{Passive Challenge stage (P C)}: The challenger \(C\) draws a bit at random \(b \leftarrow \{0, 1\}\) and runs the protocol \(\pi(T(K_b), R(k), S(k, K_b))\) in front of the adversary. Let \(T_b\) denote the transcript of this execution. \(A\) outputs a bit \(A(V_0, V_1, T_b) \rightarrow b\)
    \item \textbf{Active Challenge stage (A C)}: The challenger \(C\) draws a bit at random \(b \leftarrow \{0, 1\}\), and the adversary chooses the challenge \(c \leftarrow A\). The protocol \(\pi(T(K_b), R(c, k), S(k, K_b))\) is run in front of the adversary. Let \(T_b\) denote the transcript of this execution. Then, \(A\) outputs a bit \(A(V_0, V_1, T_b) \rightarrow b\)
\end{itemize}

We define the advantage of \(A\) in the experiment \(\text{Exp}_{\text{IBS}}^{A, \lambda, \ell}\) in a combination of modes (\textit{P QPC, P QC, A QAC, A QAC}), as the probability of outputting the same bit as the challenger in the appropriate challenge stage:

\[
\text{Adv}(A, \text{Exp}_{\text{IBS}}^{A, \lambda, \ell}) = |\Pr[b = \hat{b}] - 1/2|.
\]

The scheme \textit{IBS} is anonymous if \(\text{Adv}(A, \text{Exp}_{\text{IBS}}^{A, \lambda, \ell}) \leq \epsilon_\lambda(\lambda)\).

\section*{IV. BATCH IBS FROM REGULAR IS}
\subsection*{A. REGULAR SECURE IS}
Assume the existence of a regular IS secure in the sense of Definitions 4 and 5. Moreover, assume that \((f, D, E)\) are the chosen parameters of the scheme protocol \(\pi, s.t.\):

\begin{enumerate}
    \item \(f\) denotes an efficient function of the tag which calculates the answer \(f(k_i, c) \rightarrow r_i\) sent to the reader. We assume that the tag and the server share the same definition of \(f\). We abstract from the implementation of \(f\), however we assume that it is done in such a way which fulfills the security requirements, according to assumed definitions. This also refers to the modes of operations. E.g., if the scheme is presumably secure in the active adversary mode, this means that the function \(f\) used allows for this assumption.
    \item \(D, E\) is an effective and secure encryption scheme used in communication between the reader and the server, i.e., the reader encrypts \(c\) and \(r_i\) with a key shared with the server \(E_k(r_i, c) \rightarrow e_i\), the server decrypts \(D_k(e_i, c) \rightarrow (r_i, c)\).
\end{enumerate}

There are two general approaches to identification of the tag by the server:

\begin{enumerate}
    \item The server could use the inverse function: \(\hat{f} = f^{-1}\) to reverse the calculations done by the tag, and compute \(k_i \leftarrow \hat{f}(r_i, c)\) which would allow to effectively find and identify the tag corresponding to \(k_i\) in the database. This however requires special properties for \(f\). Note that it must be infeasible to recover one of \(f\)’s parameters (the key) given the other parameter \((c)\) and its output \((r_i)\). This, and severe constraints on tag’s computational power, makes construction of such \(f\) highly problematic. Therefore we reject this way of tag identification from further analysis.
    \item The server, having decrypted parameters \(c, r_i\), can look in the database for \(t_i\) for which the corresponding \(k_i\) gives \(r_i = f(k_i, c)\). This can be done effectively, although it requires one exhaustive search over \(\Omega\). In Alg. 1 we define a secure IS which uses this method for tag identification on a server.
\end{enumerate}

\section*{Algorithm 1 Assumed Secure IS}
\textbf{Require: }\textit{IS} = (\text{ParGen, KeyGen, } t, R, S, \pi )
\textbf{Ensure: } \text{Adv}(A, \text{Exp}_{\text{IBS}}^{A, \lambda, \ell}) \leq \epsilon_\lambda(\lambda),
\text{and } \text{Adv}(A, \text{Exp}_{\text{IBS}}^{A, \lambda, \ell}) \leq \epsilon_\lambda(\lambda)
\textbf{function Init()}
\quad (f, D, E, \text{ aux}) = \text{params} \leftarrow \text{ParGen}(1^\lambda),
\quad (k_i, k) \leftarrow \text{KeyGen()}
\textbf{end function}
\textbf{function }\pi(t(k_i), R(k), S(k, k_i))
\quad \mathcal{R} : c \leftarrow \mathcal{C}, \text{ send } c \text{ to } t,
\quad t : r_i \leftarrow f(k_i, c), \text{ send } r_i \text{ to } \mathcal{R},
\quad \mathcal{R} : e_i \leftarrow E_k(r_i, c), \text{ send } e_i \text{ to } S,
\quad S : (r_i, c) \leftarrow D_k(e_i),
\quad \text{if } r_i = f(k_i, c) \text{ then }
\quad \quad S : \text{accept}
\quad \quad \text{else}
\quad \quad \quad S : \text{reject}
\quad \text{end if}
\textbf{end function}

\subsection*{B. BATCH TAG IBS – GENERAL CONSTRUCTION}
We extend the single-tag instance into a case where an entire batch of tags participate in the protocol. Alg. 2 presents the steps executed by all parties of the system. First, \(\mathcal{R}\) generates
a random challenge $c$ from the set of all possible challenges $C$ and sends it to all tags. From the tag’s perspective, the protocol doesn’t change: each $i$-th tag from the batch $T$ calculates its response $r_i = f(k_i, c)$ as in the regular IS, and sends it back to the reader. What is different here is that, upon collecting the set of all responses $R = \{r_i\}$, the reader creates an authentication batch $B = R(c, e)$, where $g$ is some aggregation function. Then, this batch is encrypted and sent to the server to be compared with the expected value of the batch. The expected value is calculated by the server using $c$ and potential keys $k_i$.

**Algorithm 2 Batch IBS**

Require: IBS = (ParGen, KeyGen, T, R, S, π), \( T = \{t_i\} \)

function Init()
\( (f, g, D, E, aux) \leftarrow \text{params} \leftarrow \text{ParGen}(\lambda) \),
\( (K, k) \leftarrow \text{KeyGen}() \), \( K = \{k_i\} \)
end function

function \( \pi(T(K), R(k), S(k, k_i)) \)
\( R: c \leftarrow S, \text{ send } c \text{ to } T, \)
for all \( t_i \in T \) do
\( t_i; r_i \leftarrow f(k_i, c), \text{ send } r_i \text{ to } R \)
end for
\( R: \text{ Collects } R = \{r_i\}, \text{ } B \leftarrow g(R, c), \text{ } R: e \leftarrow E_k(B, c), \text{ send } e \text{ to } S, \)
\( S: (B, c) \leftarrow D_k(e), \text{ if } B = g(R, c) \text{ and } r_i = f(k_i, c) \text{ for each } r_i \in R \text{ then } S: \text{ accept } \)
else
\( S: \text{ reject } \)
end if
end function

We require that $g$ has some compression capability in that the length of $B$ is smaller than the length of all concatenated $r_i$. Also, note that in this case, the problem of finding out which $t_i$’s contributed to the batch is much harder on the server-side. The simple exhaustive search approach is not applicable here, because there are \( \binom{n}{\lambda} \) possible sets to choose from! Therefore, that function $g$ must provide additional information on the set of tags that participated in the creation of $B$ on the reader’s side. To this end, we propose to use Bloom filters, which provide a promising compression feature combined with acceptable (as we show experimentally) calculation overhead on the server, while keeping the tag and reader load at a minimum.

**C. SECURITY OF GENERAL IBS**

In this section we discuss the security of the general IBS from Alg. 2, i.e., we show it is secure in our model if the underlying regular IS is secure.

**Theorem 1:** The IBS defined as in Alg. 2 is secure in the sense of Definition 7 in a chosen “mode”, assuming IS from Alg. 1 is secure in the sense of Definition 4 in the same “mode”.

**Proof:** Let Game 0 denotes the security experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ for a regular IS, in a chosen mode $M$, for a tag with an unknown secret key $k_i$, as of Definition 4. In Game 1 we chose our own key $k$, and replace the original message $e$ with the $E_k(r_i, c)$. If the adversary rejects (or behaves differently than in Game 0), they would efficiently break the semantic security of encryption scheme $(E, D)$. In Game 2 we will maintain two experiments: the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ from Game 1, and the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ transformed from the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ of Game 1. Namely for the purpose of $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ we produce a set of additional keys $(k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n)$. The $(k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n)$ plus the unknown $k_i$ form a set of the keys $K$ for the batch $T(K)$ of IBS. Next, we proceed to the query stage of $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$. If the mode $M$ is the active one, we allow the adversary $\mathcal{A}_{\text{IBS}}$ of $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ to choose the challenges $c_j$ for each $j$-th execution and pass them to the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$. In $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ we collect the tag response $r_i = f(k_i, c)$. Then, we compute the rest of tag responses for all keys from $(k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n)$ obtaining the whole set $R = \{r_i\}_1^n$ for $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$. Next, we compute $B = g(R, c, e) = E_k(B, c)$, and provide the adversary $\mathcal{A}_{\text{IBS}}$ of $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ with $(c_j, R, e) = T_j$, i.e., the knowledge for the $j$-th execution. In this way we complete $v^t = (T_1, \ldots, T_t)$ as the view of $\mathcal{A}_{\text{IBS}}$ after the $\ell$ runs of the protocol in the query stage. In the impersonation stage we allow $\mathcal{A}_{\text{IBS}}$ to choose a tag index from the batch $T$. With the probability $\frac{1}{n}$ he chooses $i$, corresponding to $t(k_i)$, i.e., the tag with the unknown key from the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$. In this case we can proceed and give $K = (k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n)$ to the adversary. Otherwise we stop. Now, if we proceed, the adversary $\mathcal{A}(K, v^t)$ having the keys $\hat{K}$, all but $k_i$, and the view $v^t$ from the query stage, tries to impersonate the whole batch $T$, including the tag $t_i$. For the challenge of $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ we collect the challenge $c$ from the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$, and pass it to $\mathcal{A}_{\text{IBS}}$. If the adversary $\mathcal{A}_{\text{IBS}}$ answers with $R = \{r_i\}_1^n$, we collect the $r_i$ which does not correspond to any of the keys in $\hat{K}$ (we can detect it since we know all the keys of $\hat{K}$), and pass it as the answer of the adversary of the experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$. Now, if $\mathcal{A}_{\text{IBS}}$ wins its game of $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$, then its answer $r_i$ is also the correct winning answer for $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$. Thus from the probability $p$ of winning $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ in Game 2, we compute the probability $p/n - \epsilon_E$ of winning the original $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ of Game 0, where $\epsilon_E$ is a negligible probability of breaking encryption $E$ – related to the transition to Game 1. Since we assume the corresponding IS is secure, i.e., $p/n - \epsilon_E$ is negligible, thus $p$ is negligible and IBS is also secure. □

**Theorem 2:** The IBS defined as in Alg. 2 is anonymous in a chosen “mode” in the sense of Definition 8, assuming the underlying IS from Alg. 1 is anonymous in the same “mode” as of Definition 5.

**Proof:** We proceed similarly to the proof of Theorem 1. Let Game 0 denotes the security experiment $\text{Exp}_{\text{IBS}}^{\{\lambda, \ell\}}$ for a
regular IS, in a chosen mode $M$, for two tags with two different
unknown secret keys $k_0, k_1$, as of Definition 5. In Game 1
we chose our own key $k$, and replace the original message $e$
with the $E_k(r, c)$. If the adversary rejects (or behaves dif-
terently than in Game 0), they would efficiently break the
semantic security of encryption scheme $(E, D)$. In Game 2
we will maintain two experiments: the experiment $\text{Exp}_{\text{IS}}^{A,\lambda,\ell}$
from Game 1, and the experiment $\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$ transformed from
the experiment $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$ of Game 1. For the purpose of
$\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$ we generate a set of additional keys ($k_2, \ldots, k_9$)
and give them to the adversary $A_{\text{IBS}}$. The ($k_2, \ldots, k_9$) plus
the unknown $k_0$ form a set of keys $K_0$ for the batch $T(K_0)$ of IBS.
Similarly, ($k_2, \ldots, k_9$) plus the unknown $k_1$ form a set of keys
$K_1$ for the batch $T(K_1)$. Next, we proceed to the query stage of
$\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$. If the mode $M$ is the active one, we allow the adversary
$A_{\text{IBS}}$ of $\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$ to choose the challenge $c_j$ for each
$j$-th execution, and pass them to the experiment $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$.
In $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$ we collect the tag responses $r_0 = f(k_0, c), r_1 = f(k_1, c)$. Then
we compute the rest of tag responses for all keys ($k_2, \ldots, k_9$), obtaining the whole sets: $R_0 = \{r_0\} \cup \{r_1\}_{1}^{n}, R_1 = \{r_1\} \cup \{r_1\}_{1}^{n}$
for the corresponding protocols runs with $T(K_0)$, and $T(K_1)$ in the query stage of $\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$. Then we compute $B_0 = g(R_0, c), e_0 = E_k(B_0, c), B_1 = g(R_1, c), e_1 = E_k(B_1, c)$, and provide the adversary $A_{\text{IBS}}$ of $\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$
with $(c_j, R_0, B_0, e_0) = T_{\text{IBS},j}$ and $(c_j, R_1, B_1, e_1) = T_{\text{IBS},j}$, i.e.,
the knowledge for the $j$-th execution of the corresponding protocols.
In this way we complete $v'_0 = \{T_{0,1}, \ldots, T_{0,\ell}\}$ and $v'_1 = \{T_{1,1}, \ldots, T_{1,\ell}\}$ as the views of $A_{\text{IBS}}$ after the $\ell$
runs of the corresponding protocols in the query stage. In the challenge stage, if the mode $M$ is the active one, we allow the adversary
$A_{\text{IBS}}$ of $\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$ to choose the challenge $c$ and pass it to the experiment $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$.
In $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$ we collect the tag response $r_0 = f(k_0, c)$. Then we compute the rest of tag responses for all keys ($k_2, \ldots, k_9$), obtaining the whole set: $R_b = \{r_b\} \cup \{r_1\}_{1}^{n}$. Then we compute $B_b = g(R_b, c), e = E_k(B_b, c)$. In this way produce the transcript $T_b$ of protocol $\pi(T(K_i), R(c, k), S(k, K_j))$ in front of the adversary. Now, the adversary $A_{\text{IBS}}(v'_0, v'_1, T_b)$ outputs its bit $b$, which we pass as the answer of the adversary in $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$. If $A_{\text{IBS}}$ wins its game of $\text{Exp}^{A,\lambda,\ell}_{\text{IBS}}$, then its answer $b$ is also the correct winning answer for $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$. Thus, from the probability $p$ of winning $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$ in Game 2, we compute the probability $p - \epsilon_E$ of winning the original $\text{Exp}^{A,\lambda,\ell}_{\text{IS}}$ of Game 0, where $\epsilon_E$ is a negligible probability of breaking encryption $E$ – related to the transition from Game 0 to Game 1. Since we assume the corresponding IS is anonymous, i.e., $p - \epsilon_E$ is negligible, thus $p$ is negligible and IBS is also anonymous.

\section*{D. COMMUNICATION OVERHEAD – LENGTH OF THE BATCH DEFINITION}

Note that the general batch protocol from Alg. 2 has an intuitive realization. The reader simply can encrypt all the responses $r_i$ it obtains from tags. This means the function $g(R, c) \rightarrow B = (R, c)$ does not have compression feature and outputs just the list $\{r_1, \ldots, r_n\}$ of all elements in $R$, and the challenge $c$. In this scenario the length of $B$ is proportional to the number of tags in $T$, but it is straightforward to obtain all $r_i$ from $B$ and check if $r_i = g(k_i, c)$ for each $r_i \in R$. The fundamental question is whether it is possible to encode all tags identifiers in a shorter message that could be efficiently decoded by the server and enables successful identification.

The main problem with the compression of the batch message from the reader to the server is the definition of the set of RFID tags, which were the subject of identification process, and which should be verified on the server. In typical dynamic scenarios, where the subset can consists of any potential element from the predefined world of all tags (denoted by $\Omega$), the definition of the subset is by enumerating the tags identifiers (of the possible minimal length), and the length of such subset definition is proportional to the cardinality of this subset. We can also think of such a definition in the context of defining all potential subsets of $\Omega$ and assigning them unique identifiers of the minimal length. Then the identifier states for the subset definition - and its elements. Thus the length of the definition is the length of the identifier. To show this let us assume that $\Omega$ is a set of distinct elements (here possible RFID tags records on the server), s.t. $|\Omega| = z$. Let $F$ be the set of its all subsets excluding the empty set $\emptyset$. There are $2^z - 1$ distinct elements that can be drawn independently from $F$. We assume that the probability distribution for drawing is uniform on $F$. There is no regularity in our data set that could help to compress the message in even some cases. Thus we need $2^z - 1$ distinct messages to be able to encode a randomly (uniformly and independently) chosen element from $F$. In order to encode $2^z - 1$ distinct messages we need $z$ bits. The above discussion is somehow disadvantageous, i.e., it shows that in typical dynamic scenarios we would not expect drastically more compression of the batch identification messages, no matter the protocol we are trying to sort out. However, in the subsequent sections we propose a batch representation based on Bloom Filters that provide quite promising compression features with the acceptable complexity cost on the server - which is only slightly higher than in the optimal case.

\section*{V. BLOOM FILTER BASED IBS}

\subsection*{A. BLOOM FILTERS}

A common definition of the Bloom filter is a tuple:

$$B^H = \left\{ H, F, \text{Add}^F(a), \text{Query}^F(a) \right\},$$

where:

- $H$ is a set of hash functions: $H = \{H_j\}$, s.t. $H_j : \{0, 1\}^* \rightarrow \{1, \ldots, m\}$,
- $k$ is the cardinality of $H$,
- $F$ is a bit array, initially set to all-zeros,
- $m$ is the length of $F$,
- $\text{Add}^F(a)$ is an operation for adding an element $a \in \{0, 1\}^*$ into the filter, i.e., it sets $F[H_j(a)] := 1$ for each $H_j \in H$,
• Query\(^H\)(a') is an operation for testing if an element a’ belongs to \(F_A\), i.e., if for each \(\mathcal{H}_j \in H\) it holds that \(F[\mathcal{H}_j(a')] = 1\) we assume w.h.p. that a’ was added to the filter by Add\(^H\)(a’) operation.

Moreover, let \(A = \{a_1, \ldots, a_n\}\) is a set of \(n\) elements \(a_i \in \{0, 1\}^*\). By \(B_{k,m}(A)\) we denote a bloom filter in which we run Add\(^H\)(a\(_i\)) for each \(a_i \in A\). A detailed description of Bloom Filters may be found in [3], [21], however the main properties of \(B_{k,m}(A)\) that we incorporate are:

1. no “false negative” identification, i.e., if for any \(\mathcal{H}_i \in H\) it holds that \(F[\mathcal{H}_i(a)] = 0\) then \(a \notin A\);
2. the probability of “false positives” can be kept arbitrarily small by adjusting the length of the filter and the number of hash functions used, i.e., the probability of false positives is \((1 - e^{-kn/m})^k\), and
3. for fixed \(m\) and \(n\), the number of hash functions that minimizes the false positive probability is \(k = \frac{m}{n}\ln 2\).

Bloom filter is often used to improve performance and storage requirements of algorithms, e.g., it can speed up computation of Private Set Intersection (PSI) of massive datasets [24], and it could resolve scalability issues of IP-based Internet by enhancing the performance of Named Data Networking (NDN) [23].

B. SCHEME PROPOSITION

Alg. 3 presents our proposal of Bloom-based batch identification scheme as an implementation of the protocol from Alg. 2. By using as \(g\) the Bloom filter (specifically, its operation Add\()\) we obtain the required compression of the identification batch, as well as provide means for more effective discovery of \(t_i\)’s participating in the batch on the server-side, through applying Query\()\) function.

In the Bloom-based protocol, the behavior of a single tag remains the same as before: upon receiving \(c\) they calculate \(r_i\) and return it to the reader. Then, however, the reader inserts all \(r_i\)’s into a Bloom filter which is encrypted and sent to the server. The server scans its database and for all \((t_i, k_i)\) calculates \(r_i = f(k_i, c)\) and checks if Query\(^H\)(r\(_i\)) == 1.

If that is true, then w.h.p. it is assumed that \(r_i \in R\) and consequently \(t_i \in T\). Note that this operation is linear with \(z\).

Also, the use of the Bloom filter effectively compresses the authentication batch \(R\). Indeed, with \(m\) being the length of the filter, \(n\) – the number of tags scanned during the session, the overall count of bits sent during one execution of the protocol in Alg. 3 is \(m + \lceil \log_2 n \rceil\).

C. COMMUNICATION OVERHEAD

Furthermore, by the following theorem, we provide an upper bound on the length of the filter that assures an identification of all tags contained in the filter.

Theorem 3: The length of Bloom filter equal to \(m^* = 1.5 \log_2 z\) is sufficient to reduce the number of false positives to at most 1 w.h.p.

Proof: Let us compare the number of operations required in order to determine which tags are contained in the batch. Fix \(k = \frac{m}{n}\ln 2\), so that probability of false positive is minimized.

\[
m^* = \frac{-n \ln p}{\ln 2} \leq \frac{n \ln \left(\frac{z - n}{z}\right)}{\ln 2} = \frac{n \ln \left(\frac{z - n}{z}\right)}{\ln 2} \leq 1.5 \log_2 (z - n).
\]

By substituting \(m\) with the obtained bound on \(m^*\) in \(t\), we obtain \(t^* \leq 1 - \frac{\epsilon}{\ln 2}\), hence the expected number of false positives is less than 1.

Finally, the total number of operations required by our protocol to fully identify the tags in the transmitted batch is equal in the expectation to

\[
z \cdot k + \left(\frac{n + t^*}{n}\right) = z \cdot \frac{m^*}{n} \ln 2 + \left(\frac{n + t^*}{n}\right) \leq 1.5 z \ln z + n + 1.
\]

D. NUMERIC EXAMPLE

Let the cardinality of the database \(\Omega\) be \(z = 32000\). Let the number of tags in the batch be \(|T| = n = 1000\). If there is no efficient tag definitions in the message from the reader to the server then the server computation complexity is to check \(10^{130}\) potential subsets in \(\Omega\). If there is no compression property in \(g\) used for generating the identification set \(B\), i.e., all the responses \(r_i\) each of the length
at least equal to the length of the identifier $|l_i| = 96$ bits get concatenated, we have $|B| = 96000$ bits. Using the Bloom filter based protocol proposed above (with the optimal number of hash functions equal to $k = \frac{m}{n} \ln 2 \approx 15$), requires less than 499000 operations with the length of the batch $|B| < 22500$ bits.

Note that our bounds are not tight. However, even such a result requires less server computation than sending a longer Bloom filter to minimize the probability of small positive. Also, note that the suggested parameters result in the expected number of false positives less than 1, where using $n = 32000$ with the optimal $k = 22$ results in the expected number of false positives almost 0, however, it requires sending 32015 bits and at least 704000 server operations.

E. BEYOND BLOOM FILTERS

Note that the general approach to batch authentication schema proposed in Sect. V-B, and examined with Bloom filters in Sect. V, may serve as an example of a more general framework, i.e., utilizing the proposed schema with a modified data structure provides different properties. In fact, any data structure with compressing properties may be used. Below we provide examples of such modified protocols that are based on Cuckoo filters [6], [8], [22] and Enhanced double hashing [5].

1) ENHANCED DOUBLE HASHING

Enhanced double hashing is a method of calculating the value of a hash function that may be used while creating a Bloom filter. Its experimental evaluation presented in [5] shows that it is one of the methods that achieves results closest to the theoretical bounds, hence we chose this method for numerical evaluation of our protocol.

2) CUCKOO FILTER

Cuckoo filters are data structures built by merging the properties of cuckoo hash tables and Bloom filters. Cuckoo filters were introduced in [8], got significant attention, with [6] and [22] being among recent results. The structure has many interesting properties with lower space overhead in practical settings than a common Bloom filter, however, some of the properties, like deleting an item from the structure, lie outside of our scope of interest due to being unnecessary in the batch authentication protocol. A cuckoo filter $CF$ utilizes 2 hash functions and is described by the following parameters: $m$ – being the number of buckets (slots) in which the entries about the stored items are kept; $b$ – the number of entries that may be kept in a single bucket; $f$ – the length of a fingerprint of each item. The idea of the structure is as follows: for each new entry $x$ two possible indexes $i_1, i_2$ are calculated and if in at least one of them there is still a place for the new entry (i.e., they store less than $b$ entries) the entry is placed in that bucket. In the case when both buckets are full one of the buckets, say $i_1$, is selected at random, and a random entry $r$ is removed from that bucket, $x$ is stored in the emptied space and $r$ is inserted into its alternative bucket (the procedure might require repeating if $r$‘s alternative bucket is full). Note that contrary to the structure described in Sect. V-A, where the total size was equal to $m$ bits, $CF(m, b, f)$ requires $m \cdot b \cdot f$ bits of storage.

3) ANALYSIS

While for Bloom filters there are $k$ hash functions that are calculated and a single entry is represented by at most $k$ bits (recall that a single bit might contribute to the representation of multiple entries), in the cuckoo filter, each entry is represented by $f$ bits. One can easily derive that $f = \log \frac{b}{z}$ and calculate the probability of false-positive and the optimal length of the filter. Moreover, for a fair comparison with the Bloom filter we set $b = 1$.

Lemma 1: For cuckoo filter with parameters $b = 1$ and $f = \log n$, the positive false entry recognition occurs with probability $Pr[\text{false positive}] = 1 - (1 - \frac{2}{mn})^n$.

Proof: The false positive occurs if at least one entry has the same first candidate bucket, say $i_1$, as the falsely positive one and the fingerprint matches or both candidate buckets for the entries are the same (note that in this case fingerprint has to match), hence $Pr[\text{false positive}] = Pr[\text{at least 1 entry has index } i_1] \cdot Pr[\text{fingerprint matches}] + Pr[\text{fingerprint matches}] = (\frac{2}{mn}) + [1 - (\frac{2}{mn})](\frac{2}{mn}) = 1 - (1 - \frac{2}{mn})^n$.

Theorem 4: The number of buckets $m^* = \frac{2}{n(1 - (1 - \frac{2}{mn})^n)}$ is sufficient to reduce the number of false positives, for $z$ tags in the database, to at most 1 w.h.p.

Proof: Directly from Lemma 1, we derive that the expected number of false positives is equal to $(z - n)(1 - (1 - \frac{2}{mn})^n)$. As we want to limit the number to at most 1, after few transformations we obtain $m^* = \frac{2}{n(1 - (1 - \frac{2}{mn})^n)}$.

Caveat 1: To achieve at most 1 false positive, the Cuckoo filter requires a smaller size only when dealing with relatively large batches, i.e., $\frac{z}{n}$ is not negligible.

To fully compare cuckoo filters to Bloom filters we calculate the computational overhead when the batch is created.

Theorem 5: The expected number of operations while creating cuckoo filter with $m^*$ buckets of size 1 is equal to $\frac{2}{n^2}(n - 1)(1 - (1 - \frac{1}{n^2})^\frac{1}{2}) + 3n = O(n)$.

Proof: First, let us note that the expected cost of creating the filter consists of creating $2n$ hash functions (creating fingerprints may be done while calculating hash, if not one has to include additional $n$ operations), checking $n$ alternative positions and the cost of entries reallocation. Clearly the expected number of reallocation steps is equal to $\frac{n(n-1)}{2n^2}$, hence the expected number of operation equals to $3n + \frac{n(n-1)}{2n^2}$. After substituting $m = m^*$, we obtain $\frac{1}{n^2}(n - 1)(1 - (1 - \frac{1}{n^2})^\frac{1}{2}) + 3n$. 


Note that identifying the batch requires 2z hash functions and z fingerprints to be calculated and verifying at most \( \frac{1}{2m} \) alternative locations.

Caveat 2: For real life range of parameters, the cuckoo filter outperforms the Bloom filters in average case.

VI. PROOF-OF-CONCEPT IMPLEMENTATIONS

To set our proposal in a real-world perspective, we conducted a series of tests using different implementations of underlying hashing algorithms to assess the time necessary to recover the set \( T \) on the server. The most important issue during implementation is choice of the method how hash functions should be generated. We focused on four methods: One Hashing [19], Double Hashing [15], Triple Hashing [5] and Enhanced Double Hashing [5]. We created implementations for each of the methods in order to examine and compare their effectiveness. Results should be compatible with assumption related to size of Bloom Filter.

In our implementation, we choose Python in version 3.10.5, which contains built-in library hashlib. To measure the execution times, the python module timeit has been used.

We implemented several methods. One Hashing method is divided into two stages. The first stage is just using one hash function and then in the second stage (Modulo Stage), we have to compute \( h(x) \mod m_i \), where \( m_i \) is the length of the actual partition of the bit vector. The Double Hashing method is expressed by the following formula: \( f(x) = a(\delta) + i \times b(\delta) \mod m \). SHA3-256 has been used as \( a() \), while for \( b() \) we decided to use SHA256. The Triple Hashing method is represented with the following expression: \( f(x) = a(\delta) + i \times b(\delta) + \frac{(\delta - i)}{2} \times c(\delta) \mod m \). It should resolve issues with a non-zero value of \( b(\delta) \). That particular method uses three different hash functions. In our example, it was SHA3-256, SHA256 and SHA512. Function \( f(x) = a(\delta) + i \times b(\delta) + \frac{(\delta - i)}{2} \mod m \) describes the Enhanced Double Hashing method. In this approach a nonlinear indicator is used. It allows to apply only two hash functions and execution takes less time than Triple Hashing method. For that purpose we used SHA3-256 and SHA256 as the hash functions.

Our test implementation is created as a single program where the tags, the reader, and the server are simulated. Each execution was performed using a randomly chosen challenge produced by a random generator from secrets built-in library. Our tests have been performed under Windows 10 Pro, running on an Intel i7 2.7 GHz processor with 16 GB RAM. We decided to perform 10000 full protocol executions of each method for \( n = 1000, z = 32000 \) and 2000 rounds for \( n = 7000, z = 224000 \) and \( n = 25000, z = 800000 \), where size is determined as \( m = 1.5n \log_2 z \). The acquired results are presented in Table 1, 2 and 3. These tables present the number of elements the server was able to obtain from the Bloom Filter sent by the reader.

We decided to also test Cuckoo Filter although it provides additional features which will not be used, the results are presented in Tables 4, 5, and 6. Number of buckets is the same as in Bloom Filter.

### TABLE 1. Bloom Filter: \( n = 1000, z = 32000, 10000 \) rounds.

| Method              | Average | Max  |
|---------------------|---------|------|
| One Hashing         | 1000.6939 | 1005 |
| Double Hashing      | 1002.7179 | 1012 |
| Triple Hashing      | 1000.6530 | 1006 |
| Enhanced Double Hashing | 1000.6994 | 1007 |

### TABLE 2. Bloom Filter: \( n = 7000, z = 224000, 2000 \) rounds.

| Method              | Average | Max  |
|---------------------|---------|------|
| One Hashing         | 7000.5295 | 7004 |
| Double Hashing      | 7002.1935 | 7009 |
| Triple Hashing      | 7000.5275 | 7005 |
| Enhanced Double Hashing | 7000.5735 | 7005 |

### TABLE 3. Bloom Filter: \( n = 25000, z = 800000, 2000 \) rounds.

| Method              | Average | Max  |
|---------------------|---------|------|
| One Hashing         | 25000.454 | 25006 |
| Double Hashing      | 25001.869 | 25007 |
| Triple Hashing      | 25000.418 | 25004 |
| Enhanced Double Hashing | 25000.455 | 25003 |

### TABLE 4. Cuckoo Filter: \( n = 1000, z = 32000, 10000 \) rounds.

| Bucket size | Fingerprints size | Average | Max  |
|-------------|-------------------|---------|------|
| 2           | log_2 32000       | 1000.265 | 1003 |
| 3           | log_2 32000       | 1000.277 | 1004 |

### TABLE 5. Cuckoo Filter: \( n = 7000, z = 224000, 2000 \) rounds.

| Bucket size | Fingerprints size | Average | Max  |
|-------------|-------------------|---------|------|
| 2           | log_2 224000      | 7000.228 | 7003 |
| 3           | log_2 224000      | 7000.266 | 7003 |

### TABLE 6. Cuckoo Filter: \( n = 25000, z = 800000, 2000 \) rounds.

| Bucket size | Fingerprints size | Average | Max  |
|-------------|-------------------|---------|------|
| 2           | log_2 800000      | 25000.10 | 25002 |
| 3           | log_2 800000      | 25000.01 | 25002 |

### TABLE 7. Performance comparison.

| Method              | Execution time (100000 rounds) | Average | Max  |
|---------------------|---------------------------------|---------|------|
| One Hashing         | 0.06106 ms                      | 0.88895 ms |
| Triple Hashing      | 0.02476 ms                      | 0.31383 ms |
| Enhanced Double Hashing | 0.01437 ms              | 0.12342 ms |

A. INTERPRETATION

It has been shown that One Hashing, Triple Hashing, and Enhanced Double Hashing have a similar indicator of obtained elements. Results for these methods are in accordance with assumption that \( m = 1.5n \log_2 z \). Triple Hashing has a little better effectiveness indicator but it requires using an additional hash function. Before applying that method, performance should be considered. We performed tests of element addition operation to Cuckoo Filter and Bloom Filter using methods presented in Table 7. Although One Hashing
requires only one hash function, that method had the worst performance during the tests. The main reason is the fact that we need to perform a modulo operation on bigger numbers than in other methods. As it has been shown, Enhanced Double Hashing has 42% better performance than Triple Hashing. In the case of the Cuckoo Filter, it is mandatory to remember that in some scenarios additional computations are needed which significantly increases the required time [8]. We also performed tests for Bloom Filter with parameters \( n = 1000, z = 32000 \) with Enhanced Double Hashing and One Hashing to show the linear complexity of adding a given number of elements. The results have been shown in Fig. 2. The \( X \) axis denotes the number of added elements and the \( Y \) axis presents the time needed for operation. Considering the effectiveness to performance ratio, we recommend the implementation of Bloom Filter based on the Enhanced Double Hashing method.

Our experiments proved that the proposed modifications of the scheme are not only correct and secure on the theoretical level, but also can be effectively implemented and used in practice.

VII. CONCLUSION

In this paper, we presented some downsides of a solution for batch tag authentication from [4]. We presented a general method to construct a secure batch tag identification protocol from a single-tag scheme assuming that the underlying protocol is secure. We discussed its security in both passive and active adversary models. We presented a feasible batch tag protocol based on the Bloom filter. Our solution addresses the issue of an impractical number of operations that need to be performed by the authenticating server to properly identify which tags are about to be authenticated and the issue of communication overhead. Our solution allows to tweak parameters of the Bloom filter to adjust the number of bits sent to the server and the cost of computation required by the server. We compare our proposition with other techniques, namely double hashing, and cuckoo filters. In the end, we implemented solutions based on regular Bloom filters, double hashing, and cuckoo filters to test them in practice. The test shows it is indeed possible to implement our protocol and use it in real-life applications.

REFERENCES

[1] R. Baashirah and A. Abuzneid, “Survey on prominent RFID authentication protocols for passive tags,” Sensors, vol. 18, no. 10, p. 3584, Oct. 2018.
[2] P. Blaśkiewicz, L. Krzywiecki, and P. Syga, “RFID tags batch authentication revisited—Communication overhead and server computational complexity limits,” in Information Security Practice and Experience (ISPEC) (Lecture Notes in Computer Science), vol. 10060, F. Bao, L. Chen, R. Deng, and G. Wang, Eds. Cham, Switzerland: Springer, 2016.
[3] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Commun. ACM, vol. 13, no. 7, pp. 422–426, Jul. 1970.
[4] J. Chen, A. Miyaji, and C. Su, “A provable secure batch authentication scheme for EPCGen2 tags,” in Provable Security (ProSec) (Lecture Notes in Computer Science), vol. 8782, S. S. M. Chow, J. K. Liu, L. C. K. Hui, and S. M. Yiu, Eds. Cham, Switzerland: Springer, 2014.
[5] C. P. Dillingler and P. Manoliou, “Bloom filters in probabilistic verification,” in Formal Methods Computer-Aided Design (Lecture Notes in Computer Science), vol. 3312, A. J. Hu, and A. K. Martin, Eds. Austin, TX, USA: Springer, Nov. 2004, pp. 367–381.
[6] D. Epstein, M. T. Goodrich, M. Mitzenmacher, and M. R. Torres, “2–3 cuckoo filters for faster triangle listing and set intersection,” in Proc. 36th ACM SIGMOD-SIGACT-SIGAI Symp. Princ. Database Syst., E. Sallinger, J. Van den Bussche, and F. Geerts, Eds., Chicago, IL, USA, May 2017, pp. 247–260.
[7] EPC Tag Data Standards Version 1.3, GS1 EPCglobal, 2006.
[8] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, “Cuckoo filter: Practically better than Bloom,” in Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol., A. Seneviratne, C. Diot, J. Kurose, A. Chaintreau, and L. Rizzo, Eds. New York, NY, USA, Dec. 2014, pp. 75–88.
[9] W. Gong, K. Liu, X. Miao, Q. Ma, Z. Yang, and Y. Liu, “Informative counting: Fine-grained batch authentication for large-scale RFID systems,” in Proc. 14th ACM Int. Symp. Mobile ad hoc Netw. Comput. (MobiHoc), 2013, pp. 21–30.
[10] W. Gong, Y. Liu, A. Nayak, and C. Wang, “Wise counting: Fast and efficient batch authentication for large-scale RFID systems,” in Proc. 15th ACM Int. Symp. Mobile Ad Hoc Netw. Comput. (MobiHoc), J. Wu, X. Cheng, X.-Y. Li, and S. Sarkar, Eds., 2014, pp. 347–356.
[11] S. Goldwasser and S. Micali, “Probabilistic encryption & how to play men- tal poker keeping secret all partial information,” in Proc. 14th Annul. ACM Symp. Theory Comput. - STOC, New York, NY, USA, 1982, pp. 365–377.

[12] A. Khalid and F. Esposito, “Optimized cuckoo filters for efficient dis- tributed SDN and NFV applications,” in Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN), Nov. 2020, pp. 77–83.

[13] J. Kang, K. Fan, K. Zhang, X. Cheng, H. Li, and Y. Yang, “An ultra light weight and secure RFID batch authentication scheme for IoTM,” Comput. Commun., vol. 167, pp. 48–54, Feb. 2021.

[14] J. Katz and Y. Lindell, Introduction to Modern Cryptography: Principles and Protocols. Boca Raton, FL, USA: CRC Press, 2007.

[15] A. Kirsch and M. Mitzenmacher, “Less hashing, same performance: Build- ing a better Bloom filter,” in Algorithms, Y. Azar and T. Erlebach, Eds. Berl: Springer, Germany: Springer, 2006, pp. 25–47.

[16] L. Yu, Y. Liu, and J. Han, “ACTION: Breaking the privacy barrier for RFID systems,” Ad Hoc Sensor Wireless Netw., vol. 24, nos. 1–2, pp. 135–159, 2015.

[17] B. Li, W. Liu, and L. Wang, “Efficient and lightweight batch authentication for large-scale RFID systems,” IEEE Wireless Commun. Lett., vol. 8, no. 4, pp. 1272–1275, Aug. 2019.

[18] Q. Lin, L. Yang, and Y. Guo, “Proactive batch authentication: Fishing counterfeit RFID tags in muddy waters,” IEEE Internet Things J., vol. 6, no. 1, pp. 568–579, Feb. 2019.

[19] J. Liu, T. Yang, Y. Wang, H. Dai, L. Jin, H. Song, and B. Liu, “One-hashing Bloom filter,” in Proc. IEEE 23rd Int. Symp. Quality Service (IWQoS), Jun. 2015, pp. 289–298.

[20] A. Mahmoudi and M. Ahmadi, “2DCF-PIT: Two-dimensional neighbor- based cuckoo filter for pending interest table lookup in named data networking,” J. Netw. Syst. Manage., vol. 30, no. 3, pp. 1–28, Jul. 2022.

[21] M. Mitzenmacher, “Bloom filters,” in Encyclopedia of Database Systems, L. Liu and M. T. Özsu, Eds. Boston, MA, USA: Springer, 2009.

[22] M. Mitzenmacher, S. Pontarelli, and P. Reviriego, “Adaptive cuckoo filters,” in Proc. 20th Workshop Algorithm Eng. Exp., (ALENEX), R. Pagh and S. Venkatasubramanians, Eds. New Orleans, LA, USA, Jan. 2018, pp. 36–47.

[23] S. Nayak, R. Patgiri, and A. Borah, “A survey on the roles of Bloom filter in implementation of the named data networking,” Comput. Netw., vol. 196, Sep. 2021, Art. no. 108232.

[24] S. Qiu, Z. Zhang, Y. Liu, H. Yan, and Y. Cheng, “SE-PSI: Fog/cloud server-aided enhanced secure and effective private set intersection on scalable datasets with Bloom filter,” Math. Biosci. Eng., vol. 19, no. 2, pp. 1861–1876, 2021.

[25] F. Rahman and S. Ahamed, “Efficient detection of counterfeit products in large-scale RFID systems using batch authentication protocols,” Pers. Ubiquitous Comput., vol. 18, no. 1, pp. 177–188, Jan. 2014.

[26] M. S. Rahman, M. Soshi, K. Omote, and A. Miyaji, “An RFID authenti- cation protocol suitable for batch-mode authentication,” in Proc. Comput. Secur. Symp. (CSS-C), Oct. 2008, pp. 277–282.

[27] S. A. Soleymani, S. Goudarzi, M. H. Anisi, N. Kama, S. A. Ismail, A. Azmi, M. Zareei, and A. H. Abdollah, “A trust model using edge nodes and a cuckoo filter for securing VANET under the NLoS condition,” Symmetry, vol. 12, no. 4, p. 609, 2020.

[28] S. S. Moni and D. Manivannan, “A lightweight privacy-preserving V2I mutual authentication scheme using cuckoo filter in VANETs,” in Proc. IEEE 19th Annu. Consum. Commun. Netw. Conf. (CCNC), Jan. 2022, pp. 815–820.

[29] G. Tsuchik, “A family of dunes: Trivial RFID identification and authen- tication protocols,” in Privacy Enhancing Technologies, N. Borisov and P. Golle, Eds. Berlin, Germany: Springer, 2007, pp. 45–61.

[30] S. A. Weiss, S. E. Sarma, R. L. Rivest, and D. W. Engels, “Security and privacy aspects of low-cost radio frequency identification systems,” in Security in Pervasive Computing. Berlin, Germany: Springer, 2004, pp. 201–212.

[31] J. Xu and L. Dang, “An efficient RFID anonymous batch authentication protocol based on group signature,” Discrete Continuous Dyn. Syst. S, vol. 12, nos. 4–5, pp. 1489–1500, 2019.

[32] A. Yang, D. Boshoff, Q. Hu, G. P. Hancke, X. Luo, J. Weng, K. Mayes, and K. Markantonakis, “Privacy-preserving group authentication for RFID tags using bit-collision patterns,” IEEE Internet Things J., vol. 8, no. 14, pp. 11607–11620, Jul. 2021.

[33] L. Yang, J. Han, Y. Qi, and Y. Liu, “Identification-free batch authentication for RFID tags,” in Proc. 18th IEEE Int. Conf. Netw. Protocols, Oct. 2010, pp. 154–163.

[34] L. Yang, P. Peng, F. Dang, C. Wang, X.-Y. Li, and Y. Liu, “Anti- counterfeiting via federated RFID tags’ fingerprints and geometric relationships,” in Proc. IEEE Conf. Comput. Commun. (INFOCOM), Apr. 2015, pp. 1966–1974.

[35] Y. Zhu, C. Duan, X. Ding, and Z. Yang, “B-AUT: A universal architecture for batch RFID tags authentication,” in Proc. IEEE 27th Int. Conf. Parallel Distrib. Syst. (ICPADS), Dec. 2021, pp. 755–762.

PRZEMYŚLAW BLAŚKIEWICZ received the B.Eng. degree from Coventry University, the M.Sc. degree in computer networks from the Faculty of Electronics, Wroclaw University of Science and Technology, and the Ph.D. degree in computer security from WUST. He is currently an Adjunct Professor at WUST. His research interests include security of small and constrained devices, wireless telecommunication, and legal aspects of cryptography.

BARTOSZ DRZAZGA received the B.Eng. degree in computer science and the M.Sc.Eng. degree in algorithmic computer science with speciality in cryptography and computer security from the Wroclaw University of Science and Technology, Poland, in 2020 and 2021, respectively, where he is currently pursuing the Ph.D. degree in information and communication technology. His research interests include computer security, cryptography, and post-quantum isogeny-based cryptosystems.

ŁUKASZ KRZYWIECKI received the engineer’s and M.Sc. degrees in computer science from the Wroclaw University of Science and Technology (WUST), Poland, in 1997, the Ph.D. degree in computer science from WUST, in 2003, and the Habilitation degree in cryptography from Adam Mickiewicz University, Poland, in 2020. He is currently an Associate Professor at the Department of Computer Science, Faculty of Fundamental Problems of Technology, WUST. He has authored research articles concerning computer security, anonymity and privacy of group oriented cryptographic systems, ring signatures, broadcast encryption, identification, and AKE protocols. He has also participated in a number of European research projects. His current scientific research interests include secure protocols for untrusted IT architectures with unprotected randomness, PUF based solutions, and security of VANETs.

DAMIAN STYGAR received the M.Sc. degree in computer science from the Wroclaw University of Science and Technology. He is currently a Research Assistant at the Wroclaw University of Science and Technology with educational and research responsibilities. He is also an experienced Cyber Security Specialist and a Product Security Engineer at DataWalk with security clearance at national and international levels (EU Secret and NATO Secret). His research interests include computer security and cryptography.

PIOTR SYGA received the M.Sc. degree in computer science from the Faculty of Fundamental Problems of Technology, Wroclaw University of Technology, in 2010, and the Ph.D. degree in mathematical sciences in the field of computer science from the Institute of Computer Science, Polish Academy of Sciences, in 2015. He is currently a Researcher and a Lecturer at the Faculty of Fundamental Problems of Technology, Wroclaw University of Science and Technology. His research and professional interests include image processing and various aspects of privacy, including preserving privacy and information security in networks of severely constrained devices and biometrics.

***