# T-MATCH: Privacy-Preserving Item Matching for Storage-Only RFID Tags

Kaoutar Elkhiyaoui[1], Erik-Oliver Blass[2], and Refik Molva[1]

[1] Eurecom, Sophia-Antipolis, France
{kaoutar.elkhiyaoui,refik.molva}@eurecom.fr
[2] College of Computer and Information Science,
Northeastern University, Boston, MA 02115
blass@ccs.neu.edu

**Abstract.** RFID-based tag matching allows a reader $\mathtt{R}_k$ to determine whether two tags $T_i$ and $T_j$ store some attributes that jointly fulfill a boolean constraint. The challenge in designing a matching mechanism is tag privacy. While cheap tags are unable to perform any computation, matching has to be achieved without revealing the tags' attributes. In this paper, we present T-MATCH, a protocol for secure and privacy preserving RFID tag matching. T-MATCH involves a pair of tags $T_i$ and $T_j$, a reader $\mathtt{R}_k$, and a backend server $\mathtt{S}$. To ensure tag privacy against $\mathtt{R}_k$ and $\mathtt{S}$, T-MATCH employs a new technique based on secure two-party computation that prevents $\mathtt{R}_k$ and $\mathtt{S}$ from disclosing tag attributes. For tag privacy against eavesdroppers, each tag $T_i$ in T-MATCH stores an IND-CPA encryption of its attribute. Such an encryption allows $\mathtt{R}_k$ to update the state of $T_i$ by merely re-encrypting $T_i$'s ciphertext. T-MATCH targets cheap tags that cannot perform any computation, but are only required to store 150 bytes.

**Keywords:** RFID, tag matching, privacy

## 1 Introduction

One prominent application of RFID technology is the automation of safety inspections when transporting hazardous goods such as highly reactive chemicals in supply chains. Here, it is dangerous to place specific, reactive chemicals close to each other, because small leaks can already result in a threat to the life of workers managing these chemicals.

Some recent solutions to enforce safety regulations when storing or transporting chemicals in supply chains rely on equipping each chemical container with an RFID tag that stores information for identifying the chemical in the container as highlighted by EU project CoBIs [5]. Before two tags are placed next to each other, their tags are wirelessly "scanned" using an RFID reader. Each tag sends its content in cleartext to a server. The server performs chemicals' matching based on a set REF of *matching references* that it knows beforehand. Each matching reference identifies a pair of chemicals that react. Now, when two reactive chemicals are detected, the server triggers an alarm.

However, the above solution suffers from several shortcomings that may lead to security and privacy threats. The fact that tags transmit their contents in cleartext allows any malicious entity with proper wireless equipment to learn the content of a container, to infer information about reactive chemicals, and finally to track their location.

Consequently, RFID-based protocols for tag matching require a careful design taking into account both the security and the privacy threats to RFID tags and the consequences thereof on the security and safety of users managing matched items.

A privacy preserving RFID-based tag matching must fulfill two requirements: *content privacy* and *location privacy*. Content privacy assures that tag matching is performed without disclosing the content of tags, that is, the only information revealed after executing the protocol is a bit $b$ indicating whether the tags involved in the protocol execution "match" or not. Location privacy aims at preventing tracking attacks. Ideally, an adversary must not be able to distinguish between tags based on the traces of the matching protocol.

With respect to security, it is mandatory to ensure that a matching protocol is correct (almost) all the time. Namely, it is required to detect all incompatible items (reactive chemicals). This corresponds to a

*completeness* property: the protocol must always trigger an alarm when two reactive chemicals are put next to each other. Moreover, the protocol has to be efficient: an alarm is triggered only when necessary. When a match is detected by the protocol, one can safely derive that the tags involved in the protocol are attached to reactive chemicals. This second requirement corresponds to the *soundness* property of the protocol.

Note that solutions to answer the above security and privacy problems are strongly constrained by the limitations of RFID environment. While location privacy of tags can be ensured by using re-encryption techniques, content privacy against readers is more difficult to address especially when using cheap RFID read/write only tags unable to perform any computation. Traditional security and privacy solutions based on heavyweight secret matching protocols between two parties , cf., Ateniese et al. [2], Balfanz et al. [3], cannot be implemented in an RFID setting.

Accordingly, we design T-MATCH, a new tag matching protocol that involves tags $T_i$ attached to "containers" (barrels) of chemicals traveling in a supply chain, multiple readers $\mathtt{R}_k$ and a back-end server $\mathtt{S}$. T-MATCH targets read/write only tags only featuring storage and no computational capabilities. These tags are the cheapest type of tags and therefore can allow for the deployment of such an application with reasonable cost.

*Overview:* In T-MATCH, a reader $\mathtt{R}_k$ in the supply chain reads out the content of a pair of tags $T_i$ and $T_j$, cooperates with back-end server $\mathtt{S}$ to perform tag matching, and finally outputs the outcome of matching while assuring various privacy properties in the face of curious readers $\mathtt{R}_k$ and curious backend server $\mathtt{S}$.

Reader $\mathtt{R}_k$ and backend server $\mathtt{S}$ are required to evaluate securely a *boolean* function CHECK for any pair of tags $T_i$ and $T_j$, such that CHECK outputs $b = 1$, if $T_i$ and $T_j$ match. To this effect, each tag $T_i$ in T-MATCH stores a homomorphic IND-CPA encryption $\mathtt{Enc}$ of its attribute $a_{T_i}$. When two tags $T_i$ and $T_j$ are in the range of reader $\mathtt{R}_k$, reader $\mathtt{R}_k$ reads both tags and retrieves the encryptions $\mathtt{Enc}(a_{T_i})$ and $\mathtt{Enc}(a_{T_j})$ of the attributes of $T_i$ and $T_j$ respectively. To protect the privacy of tags, reader $\mathtt{R}_k$ re-encrypts the ciphertexts stored into tags $T_i$ and $T_j$. Now, to evaluate the CHECK function, reader $\mathtt{R}_k$ uses the homomorphic property of $\mathtt{Enc}$ to compute an encryption $\mathtt{Enc}(f(a_{T_i}, a_{T_j}))$ of a function $f$ of $T_i$ and $T_j$'s attributes. Then, reader $R_k$ and backend server $S$ engage in two party protocol for a *modified* privacy preserving plaintext equality test [9] to check whether $f(a_{T_i}, a_{T_j}) \in \mathtt{REF}$, where $\mathtt{REF}$ is the backend server $\mathtt{S}$'s set of matching references. If so, CHECK outputs $b = 1$; otherwise, CHECK outputs $b = 0$.

To summarize, T-MATCH's major contributions are:

- T-MATCH proposes a novel solution for item matching that targets read/write only tags. A tag $T_i$ in T-MATCH does not perform any computation, it is only required to store a state that is updated at every protocol execution by reader $\mathtt{R}_k$.
- T-MATCH is provably privacy preserving: neither reader $\mathtt{R}_k$ nor backend server $\mathtt{S}$ can disclose the content of a tag or learn its attribute. Also, T-MATCH relies on techniques of secure two-party computation to ensure the privacy of both reader $\mathtt{R}_k$ and backend server $\mathtt{S}$.
- T-MATCH is provably secure: readers $\mathtt{R}_k$ raise an alarm only when they interact with a pair of matching tags.

## 2 Preliminaries

In this section, we introduce T-MATCH's problem statement and T-MATCH's entities.

### 2.1 Problem statement

A read/write only tag $T_i$ in T-MATCH stores a state that encodes its attribute $a_{T_i}$. By solely relying on the states of any pair of tags $T_i$ and $T_j$, a reader $R_k$ has to decide whether tags $T_i$ and $T_j$ match or not.

A first solution to tackle this problem could be encrypting the state of tags. When two tags $T_i$ and $T_j$ are in the range of an authorized reader $\mathtt{R}_k$, reader $\mathtt{R}_k$ decrypts the content of tags $T_i$ and $T_j$. Finally, based on a set of matching references $\mathtt{REF}$, reader $\mathtt{R}_k$ decides whether $T_i$ and $T_j$ match or not.

However, the solution above has two limitations: **first,** if the underlying encryption is not probabilistic, tags will be sending the same ciphertexts whenever queried. This enables any eavesdropper to track tags and, consequently, enables eavesdroppers to violate tags' *location privacy*. **Second,** it does not ensure *content privacy* against reader $\mathtt{R}_k$. The solution relies on disclosing the tags' attributes to reader $\mathtt{R}_k$ in order to perform tag matching.

Although, the first limitation can be tackled by using probabilistic encryption, the second limitation is difficult to address, as tags cannot perform any computation.

We recall that our main goal is to enable reader $\mathtt{R}_k$ to perform tag matching for any pair of tags $T_i$ and $T_j$ while preserving the privacy of tags. That is, at the end of the matching protocol, a reader $\mathtt{R}_k$ only gets the outcome of a boolean function CHECK which outputs a bit $b = 1$ if tags $T_i$ and $T_j$ match, and $b = 0$ otherwise.

A straightforward solution to address the problem above is to use homomorphic encryption. Homomorphic encryption enables readers $\mathtt{R}_k$ to compute the encrypted value $\mathtt{Enc}(\text{CHECK}(T_i, T_j))$ using the encrypted value $\mathtt{Enc}(a_{T_i})$ of attribute $a_{T_i}$ stored in tag $T_i$ and the encrypted value $\mathtt{Enc}(a_{T_j})$ of attribute $a_{T_j}$ stored in tag $T_j$.

However, a limitation of this approach arises when we allow readers to decrypt $\mathtt{Enc}(\text{CHECK}(T_i, T_j))$: if a reader $\mathtt{R}_k$ is allowed to decrypt $\mathtt{Enc}(\text{CHECK}(T_i, T_j))$, then by the same means, it can decrypt $\mathtt{Enc}(a_{T_i})$ and $\mathtt{Enc}(a_{T_j})$, leading to the potential disclosure of the attributes of tags to readers and thus, violating the requirement of content privacy.

An idea to overcome this limitation, consists of preventing readers from decrypting ciphertexts by themselves. This calls for the use of secret sharing techniques [16]. We identify two methods to implement secret sharing: the **first** method relies on distributing secret shares to readers and tags. The idea would be to allow a reader $\mathtt{R}_k$ to decrypt only when it reads a pair of tags $T_i$ and $T_j$ that match. However, such a solution requires that tags $T_i$ in the system are either active and able to perform cryptographic operations, or synchronized by readers. The **second** method relies on an additional third-party component that is a backend server S. S possesses the set REF of matching references. Readers and backend server S hold secret shares of some secret key sk that allows backend server S and any reader $\mathtt{R}_k$ to evaluate securely $\text{CHECK}(T_i, T_j)$.

T-MATCH relies on the second method to implement item matching. That is, in addition to readers $\mathtt{R}_k$ which read and re-encrypt the content of tags, T-MATCH involves a backend server S that stores the set REF of matching references for any pair of attributes that match. Although, this approach requires backend server S to be always online with readers $\mathtt{R}_k$, it remains realistic. We stress that, today, even handheld RFID readers can establish continuous connection with backend server S using wireless technologies such as Bluetooth, ZigBee, WiFi or even GSM. Furthermore, having a backend server S allows for using techniques of secure multi-party computation to ensure that at the end of an execution of T-MATCH, readers $\mathtt{R}_k$ and backend server S learn at most the output of CHECK.

Now, to check whether a pair of tags $T_i$ and $T_j$ match, a reader $\mathtt{R}_k$ reads first the encrypted states stored into $T_i$ and $T_j$, then $\mathtt{R}_k$ contacts backend server S in order to securely evaluate the CHECK function for $T_i$ and $T_j$. The CHECK function has as input the encrypted states of tags $T_i$ and $T_j$ along with the backend server S's matching references REF. At the end of a T-MATCH's execution, reader $\mathtt{R}_k$ gets the output of the CHECK function. If CHECK outputs $b = 1$, then this implies that $T_i$ and $T_j$ match; otherwise, they do not.

### 2.2 T-MATCH's Setup

T-MATCH involves the following entities:

- **Tags $T_i$:** Each tag is attached to an item (container, barrel, ...). A tag $T_i$ is equipped with a re-writable memory storing $T_i$'s current "state" denoted $s^j_{T_i}$. The state $s^j_{T_i}$ encodes an attribute $a_{T_i} \in \mathbb{A}$, where $\mathbb{A}$ is the set of valid attributes in T-MATCH. We denote $\mathcal{T}$ the set of tags in T-MATCH, $|\mathbb{A}| = l$ and $|\mathcal{T}| = n$.
- **Issuer I:** The issuer I initializes tags. It chooses an attribute $a_{T_i} \in \mathbb{A}$, then computes an initial state $s^0_{T_i}$, and finally writes the state $s^0_{T_i}$ into $T_i$.
- **Readers $\mathtt{R}_k$:** A reader $\mathtt{R}_k$ interacts with tags $T_i$ in its vicinity. $\mathtt{R}_k$ reads the states $s^{k_i}_{T_i}$ and $s^{k_j}_{T_j}$ stored into tags $T_i$ and $T_j$ respectively by calling the function READ, and updates the states $s^{k_i}_{T_i}$ and $s^{k_j}_{T_j}$ accordingly. Next, $\mathtt{R}_k$ writes the new states $s^{k_i+1}_{T_i}$ and $s^{k_j+1}_{T_j}$ into $T_i$ and $T_j$ by calling the function WRITE. Finally, $\mathtt{R}_k$ engages in a *two party protocol* with backend server S to compute securely a boolean function CHECK. $\mathtt{R}_k$'s input to CHECK is the states $s^{k_i}_{T_i}$ and $s^{k_j}_{T_j}$. If CHECK outputs $b = 1$, then reader $\mathtt{R}_k$ raises an alarm meaning that $T_i$ and $T_j$ match. Otherwise, $T_i$ and $T_j$ do not match and reader $\mathtt{R}_k$ does nothing. Without loss of generality, we assume that T-MATCH comprises $\eta$ readers $\mathtt{R}_k$.
- **Backend server S:** Backend server S stores a set of $\nu$ matching references $\mathtt{REF} = \{\mathtt{Ref}_1, \mathtt{Ref}_2, ..., \mathtt{Ref}_\nu\}$. Backend server S is required to compute a boolean function CHECK jointly with reader $\mathtt{R}_k$. Backend server S's input to the CHECK function is its set of matching references REF.

## 3 Adversary models

In this paper, we only focus on *semi-honest* adversaries. That is, readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ are assumed to act according to the protocol with the exception that each party keeps a record of all its computations. Note that in the real world, it is hard for readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ to deviate from the protocol arbitrarily without being detected. In effect, it is always feasible to verify whether a reader $\mathtt{R}_k$ raises an alarm when it should or not. Whereas, it is hard to prevent readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ from keeping records of their previous protocol executions or from eavesdropping on tags in the system.

In line with previous work on secure multiparty computation [6], we define security and privacy of T-MATCH by considering a computation of CHECK in the *ideal* model with a trusted third party (TTP) and an execution of T-MATCH in the *real* model without a TTP. In the ideal model, a TTP gets the private input of backend server $\mathtt{S}$ and the input of a reader $\mathtt{R}_k$ which is composed of the states of a pair of tags $T_i$ and $T_j$ participating in tag matching and that were read by reader $R_k$, together with $\mathtt{R}_k$'s private input. Then, the TTP outputs the outcome of $\text{CHECK}(T_i, T_j)$.

In brief, T-MATCH is said to be secure and privacy preserving, if the malfunctioning which may occur when executing T-MATCH in the real model cannot be avoided when a trusted third party computes the output of CHECK.

In the rest of the paper we assume that:

**i.)** readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ are semi-honest;

**ii.)** issuer $I$ is honest;

**iii.)** multiple readers $\mathtt{R}_k$ can collude against tags. However, readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ do not collude against tags. Otherwise, tag privacy cannot be achieved.

Now, to formally capture the capabilities of an adversary $\mathcal{A}$ against T-MATCH, we allow $\mathcal{A}$ to access the following oracles:

- $\mathcal{O}_{\mathtt{Tag}}(\mathtt{param})$: When queried with a parameter $\mathtt{param} \in \mathcal{T} \cup \mathbb{A}$, the oracle $\mathcal{O}_{\mathtt{Tag}}(\mathtt{param})$ returns a tag based on the value of the parameter chosen by $\mathcal{A}$. If $\mathtt{param} = T_i \in \mathcal{T}$, then $\mathcal{O}_{\mathtt{Tag}}(\mathtt{param})$ returns tag $T_i$. If $\mathtt{param} = a_i \in \mathbb{A}$, then $\mathcal{O}_{\mathtt{Tag}}(\mathtt{param})$ returns a tag that encodes attribute $a_i$.
- $\mathcal{O}_{\mathtt{Check}}(T_i, T_j)$: When queried with a pair of tags $T_i$ and $T_j$, the oracle $\mathcal{O}_{\mathtt{Check}}$ returns a bit $b = \text{CHECK}(T_i, T_j)$. If $b = 1$, then this entails that $T_i$ and $T_j$ store a pair of attributes that match; otherwise, they do not.
- $\mathcal{O}_{\mathtt{CorruptR}}(\mathtt{R}_k)$: When queried with a reader $\mathtt{R}_k$, the oracle $\mathcal{O}_{\mathtt{CorruptR}}$ returns $\mathtt{R}_k$'s secret information denoted $\mathtt{Sec}_k$. We say that reader $\mathtt{R}_k$ is controlled by $\mathcal{A}$.
- $\mathcal{O}_{\mathtt{CorruptS}}$: When queried, the oracle $\mathcal{O}_{\mathtt{CorruptS}}$ returns $\mathtt{S}$'s secret information denoted $\mathtt{Sec}_\mathtt{S}$. We say that server $\mathtt{S}$ is controlled by $\mathcal{A}$.
- $\mathcal{O}_{\mathtt{Flip}}(\mathtt{T}_0, \mathtt{T}_1)$: When queried with two tags $\mathtt{T}_0$ and $\mathtt{T}_1$, $\mathcal{O}_{\mathtt{Flip}}$ flips a fair coin $b \in \{0, 1\}$. If $b = 1$, $\mathcal{O}_{\mathtt{Flip}}$ returns tag $\mathtt{T}_1$; otherwise, it returns tag $\mathtt{T}_0$.

### 3.1 Security

In the following, we introduce the security requirements of T-MATCH against an adversary $\mathcal{A}$.

**Completeness** Completeness ensures that if two tags $T_i$ and $T_j$ store a pair of matching attributes, then $\text{CHECK}(T_i, T_j)$ outputs $b = 1$.

**Definition 1.** T-MATCH *is complete* $\Leftrightarrow$ *For any pair of tags* $(T_i, T_j)$ *that store a pair of matching attributes,* $\text{CHECK}(T_i, T_j) = 1$.

**Soundness** Soundness assures that if the CHECK function outputs $b = 1$, then this entails that the tags $T_i$ and $T_j$ presented to reader $\mathtt{R}_k$ encode a pair of attributes $a_{T_i}$ and $a_{T_j}$ that match with an overwhelming probability.

We formalize soundness using an experiment-based definition as depicted in Algorithm 1 and Algorithm 2. Adversary $\mathcal{A}$ has access to T-MATCH in two phases. In the learning phase, T-MATCH calls the oracle $\mathcal{O}_{\mathtt{Tag}}$ that supplies $\mathcal{A}$ with $r$ tags $T_i$. $\mathcal{A}$ is allowed to read and write into tags $T_i$. He can also query the oracle $\mathcal{O}_{\mathtt{Check}}$ with any tag from the set of $r$ tags $T_i$ for a maximum of $s$ times.

**for** $i := 1$ **to** $r$ **do**
  $T_i \leftarrow \mathcal{O}_{\mathtt{Tag}}(\mathtt{param}_i)$;
  **for** $j := 1$ **to** $s$ **do**
    $s_i^j = \text{READ}(T_i)$;
    $\text{WRITE}(T_i, s'^j_i)$;
    $T_{(i,j)} \leftarrow \mathcal{O}_{\mathtt{Tag}}(\mathtt{param}_{(i,j)})$;
    $s_{(i,j)} = \text{READ}(T_{(i,j)})$;
    $\text{WRITE}(T_{(i,j)}, s'_{(i,j)})$;
    $b_{(i,j)} \leftarrow \mathcal{O}_{\mathtt{Check}}(T_i, T_{(i,j)})$;
  **end**
**end**

**Algorithm 1:** $\mathcal{A}$'s security learning phase

$(\mathtt{T}_0, \mathtt{T}_1) \leftarrow \mathcal{A}$;
$b \leftarrow \mathcal{O}_{\mathtt{Check}}(\mathtt{T}_0, \mathtt{T}_1)$;

**Algorithm 2:** $\mathcal{A}$'s security challenge phase

In the challenge phase, $\mathcal{A}$ generates two tags $\mathtt{T}_0$ and $\mathtt{T}_1$. Then, $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathtt{Check}}$ with tags $\mathtt{T}_0$ and $\mathtt{T}_1$. Finally, $\mathcal{O}_{\mathtt{Check}}$ outputs a bit $b$.

$\mathcal{A}$ is said to be successful, if **i.)** $b = 1$ and if **ii.)** $\mathtt{T}_0$ and $\mathtt{T}_1$ encode two attributes $a_{\mathtt{T}_0}$ and $a_{\mathtt{T}_1}$ that do not match.

The experiment above captures the capabilities of an active adversary $\mathcal{A}$, who in addition to being able to read tags, can re-write their internal states. The adversarial goal of $\mathcal{A}$ is to provide a pair of tags $\mathtt{T}_0$ and $\mathtt{T}_1$ which do not store matching attributes, yet $\text{CHECK}(\mathtt{T}_0, \mathtt{T}_1)$ outputs 1.

**Definition 2.** T-MATCH *is sound* $\Leftrightarrow$ *For any adversary* $\mathcal{A}$, $Pr(\mathcal{A}$ *is successful*$) \leq \epsilon$, *such that* $\epsilon$ *is negligible.*

### 3.2 Privacy

In general, T-MATCH is said to be privacy preserving, if the only information learned by an adversary $\mathcal{A}$ after executing T-MATCH with a pair of tags $T_i$ and $T_j$ is the output of $\text{CHECK}(T_i, T_j)$. That is, an adversary $\mathcal{A}$ only learns whether tags $T_i$ and $T_j$ match or not.

**Privacy of readers $\mathtt{R}_k$ and backend server $\mathtt{S}$** In accordance with previous work on secure two-party computation [6], we define reader $\mathtt{R}_k$ and backend server $\mathtt{S}$ privacy in the semi-honest model by considering, first, an ideal model in which both parties communicate their inputs to a TTP that computes the output of the CHECK function for reader $\mathtt{R}_k$ and backend server $\mathtt{S}$. Then, we consider an execution of T-MATCH which evaluates the CHECK function in the real model without a TTP.

T-MATCH is said to be privacy preserving in the semi-honest model, if for every semi-honest behavior of one of the parties (reader $\mathtt{R}_k$ or backend server $\mathtt{S}$), the joint view of both parties can be simulated by a computation of the CHECK function in the ideal model, where also one party is semi-honest and the other is honest. That is, T-MATCH does not leak information about the private inputs of readers $R_k$ and backend server $\mathtt{S}$.

**Definition 3 (Privacy of reader $R_k$ and backend server $\mathtt{S}$ [6]).**

- *Let* $\bar{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an admissible pair representing adversarial behavior by reader* $\mathtt{R}_k$ *and backend server* $\mathtt{S}$ *in the real model. Such a pair is admissible if at least one party* $\mathcal{A}_i$ *is honest.*
  - *On input pair* $(X, Y)$ *(*$X$ *is* $\mathtt{R}_k$*'s input and* $Y$ *is* $\mathtt{S}$*'s input), let* $\mathtt{View}_1 = (X, r, M_1, ..., M_p, \text{CHECK}(X, Y))$ *denote the view of reader* $\mathtt{R}_k$*, where* $r$ *is the outcome of* $\mathtt{R}_k$*'s internal randomness, and* $M_i$ *is the* $i^{\mathtt{th}}$ *message that* $\mathtt{R}_k$ *has received.*
  - *Let* $\mathtt{View}_2 = (Y, r', M'_1, ..., M'_q, \perp)$ *denote the view of backend server* $\mathtt{S}$*, where* $r'$ *is the outcome of* $\mathtt{S}$*'s internal randomness, and* $M'_i$ *is the* $i^{\mathtt{th}}$ *message that* $\mathtt{S}$ *has received.*

  *We denote the joint execution under* $\bar{\mathcal{A}}$ *in the real model on input pair* $(X, Y)$ $\mathtt{Real}_{\bar{\mathcal{A}}}(X, Y)$*, and it is defined as* $(\mathcal{A}_1(\mathtt{View}_1), \mathcal{A}_2(\mathtt{View}_2))$.
- *Let* $\bar{\mathcal{B}} = (\mathcal{B}_1, \mathcal{B}_2)$ *be an admissible pair representing adversarial behavior by reader* $\mathtt{R}_k$ *and backend server* $\mathtt{S}$ *in the ideal model.*

  *We denote the joint execution under* $\bar{\mathcal{B}}$ *in the ideal model on input pair* $(X, Y)$ $\mathtt{Ideal}_{\bar{\mathcal{B}}}(X, Y)$*, and it is defined as* $(\mathcal{B}_1(X, \text{CHECK}(X, Y)), \mathcal{B}_2(Y, \perp))$.

T-MATCH *is said to be privacy preserving with respect to reader* $\mathtt{R}_k$ *and backend server* $\mathtt{S}$ *in the semi-honest model, if there is a transformation of pairs of admissible adversaries* $\bar{\mathcal{A}} = (\mathcal{A}_1, \mathcal{A}_2)$ *in the real model, into pairs of admissible adversaries* $\bar{\mathcal{B}} = (\mathcal{B}_1, \mathcal{B}_2)$ *in the ideal model, so that the distributions* $\{\mathtt{Real}_{\bar{\mathcal{B}}}(X, Y)\}_{X,Y}$ *and* $\{\mathtt{Ideal}_{\bar{\mathcal{B}}}(X, Y)\}_{X,Y}$ *are computationally indistinguishable.*

**Tag privacy** Ideally, a privacy preserving protocol for tag matching against an adversary $\mathcal{A}$ (readers $\mathtt{R}_k$ or backend sever $\mathtt{S}$) should ensure the following:

- It must be computationally infeasible for adversary $\mathcal{A}$ to tell whether two tags $T_i$ and $T_j$ store the same attribute, i.e., $a_{T_i} = a_{T_j}$. This property is called hereafter *attribute unlinkability*. This requirement is related to the content privacy of tags.
- It must be computationally infeasible for adversary $\mathcal{A}$ to tell two tags $T_i$ and $T_j$ apart. We call this requirement *tag unlinkability* in accordance with related work on RFID tag privacy, see Ateniese et al. [1], Juels and Weis [10]. This property ensures location privacy of tags participating in T-MATCH.

However, any adversary $\mathcal{A}$ who has access to the output of the CHECK function can mount a trivial attack against attribute unlinkability and tag unlinkability in both the ideal model and the real model.

To break attribute unlinkability for a pair of tags $(T_i, T_j)$, all $\mathcal{A}$ has to do is to run T-MATCH, first with pair of tags $(T_i, T_k)$ and then with pair of tag $(T_j, T_k)$. Next, if $\text{CHECK}(T_i, T_k) \neq \text{CHECK}(T_j, T_k)$, then $\mathcal{A}$ concludes that $T_i$ and $T_j$ encode different attributes and thus, $\mathcal{A}$ breaks attribute unlinkability. By the same token, adversary $\mathcal{A}$ concludes that $T_i$ and $T_j$ are different tags which breaks tag unlinkability.

Furthermore, it is impossible to ensure *tag unlinkability* against an adversary who monitors all of $T_i$'s interactions. We note that a tag $T_i$ in T-MATCH relies on readers $\mathtt{R}_k$ to update its state, and therefore, $T_i$'s state does not change between two protocol executions either in the ideal model or the real model. Accordingly, we relax the definition of tag unlinkability, by assuming that there is at least one unobserved interaction between tag $T_i$ and an honest reader $\mathtt{R}_k$ outside the range of adversary $\mathcal{A}$. This is in compliance with previous work on read/write only tags, cf., Ateniese et al. [1], Sadeghi et al. [14].

Now, we define tag privacy with respect to an adversary $\mathcal{A}$ who does not always access the CHECK function (i.e., an adversary $\mathcal{A}$ who has access to tags, however, no reader $\mathtt{R}_k$ is closeby) and who does not monitor all of the tags' interactions with readers $\mathtt{R}_k$.

**Attribute unlinkability.** It is noteworthy that if an adversary $\mathcal{A}$ breaks attribute unlinkability, then this adversary $\mathcal{A}$ can break tag unlinkability. Such an adversary can always tell two tags $T_i$ and $T_j$ apart as long as $T_i$ and $T_j$ encode different attributes, i.e., $a_{T_i} \neq a_{T_j}$. Therefore, if T-MATCH ensures tag unlinkability, it ensures as well attribute unlinkability. However, in the case where all tags in T-MATCH encode the same attribute, tag unlinkability does not imply attribute unlinkability. There might be an adversary $\mathcal{A}$ who cannot break tag unlinkability, yet adversary $\mathcal{A}$ knows that all tags encode the same attribute, and breaking thus, attribute unlinkability. Nonetheless, if all tags in the system encode the same attribute, then a tag matching application becomes obsolete. As a result, in the sequel of this paper, we assume that T-MATCH comprises at least two different attributes $a_i$ and $a_j$, i.e., $\mathbb{A} \geq 2$, and we only focus on tag unlinkability.

**Tag unlinkability.** Roughly speaking, tag unlinkability against an adversary $\mathcal{A}$ ensures that if **1.)** tags $T_i$ and $T_j$ interact with an honest reader outside the range of adversary $\mathcal{A}$ at least once, and if **2.)** $\mathcal{A}$ does not have access to the output of the CHECK function, then it is computationally infeasible for adversary $\mathcal{A}$ to distinguish between tags $T_i$ and $T_j$.

In accordance with previous work [1, 14], we use an experiment based definition to formalize tag unlinkability.

In the learning phase as depicted in Algorithm 3, $\mathcal{A}$ can call the oracle $\mathcal{O}_{\mathtt{CorruptR}}$ up to $r$ times which supplies $\mathcal{A}$ with the secret information of readers $\mathtt{R}_k$ denoted $\mathtt{Sec}_k$.

$\mathcal{A}$ can also call the oracle $\mathcal{O}_{\mathtt{CorruptS}}$ which supplies $\mathcal{A}$ with backend server $\mathtt{S}$'s secret information $\mathtt{Sec}_\mathtt{S}$. $\mathcal{A}$ is as well allowed to query the oracle $\mathcal{O}_{\mathtt{Tag}}$ which provides T-MATCH with $s$ tags $T_i$ that he can read from and write into. $\mathcal{A}$ can query the oracle $\mathcal{O}_{\mathtt{Check}}$ with any tag from the set of the $s$ tags $T_i$ for a maximum of $t$ times.

In the challenge phase, cf. Algorithm 4, $\mathcal{A}$ generates two challenge tags $\mathtt{T}_0$ and $\mathtt{T}_1$. These two tags are read outside the range of adversary $\mathcal{A}$, then they are submitted to the oracle $\mathcal{O}_{\mathtt{Flip}}$. Next, the oracle $\mathcal{O}_{\mathtt{Flip}}$ supplies $\mathcal{A}$ with tag $\mathtt{T}_b$, $b \in \{0, 1\}$. Finally, $\mathcal{A}$ outputs his guess $b'$ of the value of $b$.

$\mathcal{A}$ is said to be successful if **i.)** $b = b'$, **ii.)** $\mathcal{A}$ did not call both oracles $\mathcal{O}_{\mathtt{CorruptR}}$ and $\mathcal{O}_{\mathtt{CorruptS}}$.

We note that if $\mathcal{A}$ is allowed to corrupt both readers $\mathtt{R}_k$ and backend server $\mathtt{S}$, then any tag matching protocol based on storage only tags cannot ensure tag privacy against such an adversary. $\mathcal{A}$ can use $\mathtt{R}_k$ and $\mathtt{S}$'s secrets to get always the output of the CHECK function for any pair of tags and then uses the output of CHECK to break tag unlinkability.

**Definition 4.** T-MATCH *provides tag unlinkability* $\Leftrightarrow$ *For any adversary* $\mathcal{A}$, *Pr*($\mathcal{A}$ *is successful*) $- \frac{1}{2} \leq +\epsilon$, *such that* $\epsilon$ *is negligible.*

**for** $i := 1$ **to** $r$ **do**
  $\mathtt{Sec}_k \leftarrow \mathcal{O}_{\mathtt{CorruptR}}(\mathtt{R}_k)$ ;
**end**
$\mathtt{Sec}_\mathtt{S} \leftarrow \mathcal{O}_{\mathtt{CorruptS}}$ ;
**for** $i := 1$ **to** $s$ **do**
  $T_i \leftarrow \mathcal{O}_{\mathtt{Tag}}(\mathtt{param}_i)$;
  **for** $j := 1$ **to** $t$ **do**
    $s_i^j = \textsc{Read}(T_i)$;
    $\textsc{Write}(T_i, s'^j_i)$;
    $T_{(i,j)} \leftarrow \mathcal{O}_{\mathtt{Tag}}(\mathtt{param}_{(i,j)})$;
    $s_{(i,j)} = \textsc{Read}(T_{(i,j)})$;
    $\textsc{Write}(T_{(i,j)}, s'_{(i,j)})$;
    $\textsc{Check}(T_i, T_{(i,j)})$;
  **end**
**end**

**Algorithm 3:** $\mathcal{A}$'s tag unlinkability learning phase

$(\mathtt{T}_0, \mathtt{T}_1) \leftarrow \mathcal{A}$;
// $\mathtt{T}_0$ and $\mathtt{T}_1$ are read outside the the range of $\mathcal{A}$ by an honest reader;
$\mathtt{T}_b \leftarrow \mathcal{O}_{\mathtt{Flip}}(\mathtt{T}_0, \mathtt{T}_1)$;
$\textsc{Read}(\mathtt{T}_b)$;
$\textsc{Output}\ b'$;

**Algorithm 4:** $\mathcal{A}$'s tag unlinkability challenge phase

# 4 Protocol

To perform tag matching in T-MATCH, we store into each tag $T_i$ an IND-CPA homomorphic encryption $\mathtt{Enc}(a_{T_i})$. When reader $\mathtt{R}_k$ reads a pair of tags $T_i$ and $T_j$, it uses the homomorphic property of $\mathtt{Enc}$ to compute an encryption $C_{(i,j)}$ of a function $f$ of $T_i$ and $T_j$'s attribute, i.e., $C_{(i,j)} = \mathtt{Enc}(f(a_{T_i}, a_{T_j}))$.

Now, the matching reference of any pair of attributes $(a_i, a_j)$ is computed as $\mathtt{Ref}_{(i,j)} = f(a_i, a_j)$. To evaluate the CHECK function, reader $\mathtt{R}_k$ and backend server $\mathtt{S}$ rely on a two party privacy preserving *plaintext equality test* (PET for short) to decide whether $C_{(i,j)}$ encrypts one of $\mathtt{S}$'s matching references or not.

Although, it may seem that any IND-CPA homomorphic encryption such as Elgamal or Paillier could suit the privacy and the security requirements of T-MATCH when both reader $\mathtt{R}$ and backend server $\mathtt{S}$ are semi-honest, not all of them prevent backend server $\mathtt{S}$ from forging new matching references from its initial set $\mathtt{REF}$. We recall that Elgamal is multiplicatively homomorphic and thus the function $f$ is going to be expressed as $f(a_i, a_j) = \psi(a_i)\psi(a_j) = \mathtt{Ref}_{(i,j)}$, where $\psi$ is an encoding of attributes. We note also that Paillier is additively homomorphic, and as a consequence: $f(a_i, a_j) = \psi(a_i) + \psi(a_j) = \mathtt{Ref}_{(i,j)}$.

Therefore, neither the use of Elgamal nor Paillier as the underlying encryption technique of T-MATCH can prevent backend sever $\mathtt{S}$ from forging a new matching reference $\mathtt{ref}$ from its set $\mathtt{REF}$.

To prevent forgery of matching references, we use Boneh-Goh-Nissim (BGN) encryption. In addition to being multiplicatively homomorphic, BGN encryption allows computing an encryption of a bilinear pairing of two plaintexts from their ciphertexts. Consequently, a matching reference of two attributes $a_i$ and $a_j$ in T-MATCH is computed as: $\mathtt{Ref}_{(i,j)} = f(a_i, a_j) = f(a_j, a_i) = e(\psi(a_i), \psi(a_j))$, where $\psi$ is the attribute encoding in T-MATCH. Note that in this case, forging a new matching reference $\mathtt{ref}$ from $\mathtt{REF}$ is as hard as the Computational Diffie-Hellman problem.

Now, we introduce the definitions and the assumptions that will be used throughout the paper.

## 4.1 Tools

T-MATCH makes use of subgroups of finite composite order that support bilinear pairings as in previous work of Boneh et al. [4], Katz et al. [11].

**Bilinear pairings** Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups, such that $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of the same finite order $N$. A pairing $e$: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear pairing if:

1. $e$ is *bilinear*: $\forall\, x, y \in \mathbb{Z}_N, g, h \in \mathbb{G}$, $e(g^x, h^y) = e(g, h)^{xy}$;
2. $e$ is *computable*: there is an efficient algorithm to compute $e(g, h)$ for any $(g, h) \in \mathbb{G}^2$;
3. $e$ is *non-degenerate*: if $g$ is a generator of $\mathbb{G}$, then $e(g, g)$ is a generator of $\mathbb{G}_T$.

**Boneh-Goh-Nissim (BGN) cryptosystem** We now describe Boneh-Goh-Nissim (BGN) cryptosystem that will be used to encrypt the tags' attributes in T-MATCH.

– ***Key generation*:** On input of a security parameter $\tau$, the system obtains a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_T, e)$ such that:

1. $q_1$ and $q_2$ are two random primes. Typically, $|q_1| = |q_2| = 512$ bits.
2. $\mathbb{G}$ is a bilinear group of composite order $N = q_1 q_2$.
3. $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear pairing.

The system then picks up two random generators $g, u \in \mathbb{G}$ and sets $h_1 = u^{q_2}$. Finally, the system outputs the public key $\mathtt{pk} = (N, \mathbb{G}, \mathbb{G}_T, e, g, h_1)$ and the secret key $\mathtt{sk} = q_1$.

– ***Encryption*:** The encryption algorithm is defined in both groups $\mathbb{G}$ and $\mathbb{G}_T$.
  - *Encryption in* $\mathbb{G}$: On input of a message $m \in \mathbb{G}$, the encryption algorithm selects a random number $r \in \mathbb{Z}_N$ and computes $c = \mathtt{Enc}_{\mathbb{G}}(m) = m h_1^r$.
  - *Encryption in* $\mathbb{G}_T$: On input of a message $M \in \mathbb{G}_T$, the encryption algorithm picks a random number $r \in \mathbb{Z}_N$ and computes $C = \mathtt{Enc}_{\mathbb{G}_T}(M) = M \cdot e(g, h_1)^r \in \mathbb{G}_T$.

– ***Decryption*:** Decryption in BGN relies on computing discrete logarithm in a finite group of order $N$. Thus, decryption takes sublinear time $O(\sqrt{N})$ and, consequently, BGN is only suitable for the encryption of short messages. However, in T-MATCH we do not decrypt any ciphertext $C$. Only for sake of completeness, we detail below the decryption algorithm of BGN.
  - *Decryption in* $\mathbb{G}$: On input of a ciphertext $c \in \mathbb{G}$ and secret key $\mathtt{sk} = q_1$, the decryption algorithm computes: $\mathcal{C} = c^{q_1} = m^{q_1} \cdot h_1^{r q_1}$. Note that the order of $h_1$ is $q_1$, hence $\mathcal{C} = m^{q_1}$.
    Since $g$ is a generator of $\mathbb{G}$, then there exists $x_m \in \mathbb{Z}_N$ such that: $m = g^{x_m}$. Thus, $\mathcal{C} = (g^{q_1})^{x_m}$ and $x_m$ is computed as $\mathtt{log}_{g^{q_1}}(\mathcal{C})$. Hence, $\mathtt{Dec}_{\mathbb{G}}(c) = g^{x_m} = m$.
  - *Decryption in* $\mathbb{G}_T$: On input of a ciphertext $C \in \mathbb{G}_T$ and secret key $\mathtt{sk} = q_1$, the decryption algorithm computes: $\mathcal{C} = C^{q_1} = M^{q_1} \cdot e(g, h_1)^{r q_1} = M^{q_1}$, as the order of $e(g, h_1)$ is $q_1$.
    As $e(g, g)$ is a generator of $\mathbb{G}_T$, then there exists $x_M \in \mathbb{Z}_N$ such that: $M = e(g, g)^{x_M}$. Therefore, $\mathcal{C} = (e(g, g)^{q_1})^{x_M}$ and $x_M$ is computed as $\mathtt{log}_{e(g,g)^{q_1}}(\mathcal{C})$. Finally, $\mathtt{Dec}_{\mathbb{G}_T}(C) = e(g, g)^{x_M} = M$.

The BGN cryptosystem is semantically secure under the subgroup decision assumption. Moreover, the following homomorphic properties hold:

$$\forall\, m_1, m_2 \in \mathbb{G},\ \mathtt{Enc}_{\mathbb{G}}(m_1)\mathtt{Enc}_{\mathbb{G}}(m_2) = \mathtt{Enc}_{\mathbb{G}}(m_1 m_2)$$

$$e(\mathtt{Enc}_{\mathbb{G}}(m_1), \mathtt{Enc}_{\mathbb{G}}(m_2)) = \mathtt{Enc}_{\mathbb{G}_T}(e(m_1, m_2))$$

**Definition 5 (The subgroup decision assumption [4, 12]).** *Let $\mathcal{G}$ be a group of order $N$ where $N = q_1 q_2$ is the product of two primes $q_1$ and $q_2$. The subgroup decision assumption is said to hold in $\mathcal{G}$, if given a random element $u$ in $\mathcal{G}$, it is computationally hard to decide whether $u$ is in the subgroup of $\mathcal{G}$ of order $q_1$ or not.*

**Attribute encoding** Let $\mathbb{G}$ be a bilinear group of composite order $N = q_1 q_2$ and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear pairing.

We denote $\mathbb{G}_1$ and $\mathbb{G}_2$ the subgroups of $\mathbb{G}$ of order $q_1$ and $q_2$ respectively.

We also denote $\mathbb{G}_{T_1}$ and $\mathbb{G}_{T_2}$ the subgroups of $\mathbb{G}_T$ of order $q_1$ and $q_2$ respectively.

Let $g, u$ be two random generators of $\mathbb{G}$. By construction, $h_1 = u^{q_2}$ is a generator of $\mathbb{G}_1$ and $h_2 = g^{q_1}$ is a generator of $\mathbb{G}_2$.

Let $x_{\mathtt{I}} = q_1 x'_{\mathtt{I}}$ be the issuer's secret key, where $x'_{\mathtt{I}}$ is randomly selected in $\mathbb{Z}_N^*$.

An attribute $a_i$ in T-MATCH is encoded as $\psi(a_i) = H(a_i)^{x_{\mathtt{I}}}$, where $H : \mathbb{Z}_N \rightarrow \mathbb{G}$ is a cryptographic hash function.

To evaluate $H$, issuer $\mathtt{I}$ is provided with $g$ generator of $\mathbb{G}$ and with a cryptographic hash function $h : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$. For all $a_i \in \mathbb{Z}_N$, issuer $\mathtt{I}$ computes $h(a_i) = x_i$ and then outputs $H(a_i) = g^{x_i} \in \mathbb{G}$.

We note that, for all $a_i \in \mathbb{A},\ \psi(a_i) \in \mathbb{G}_2$:

$$\begin{aligned}\psi(a_i) = H(a_i)^{x_{\mathtt{I}}} &= (g^{x_i})^{x_{\mathtt{I}}} = g^{x_i x_{\mathtt{I}}} \\ &= g^{x_i q_1 x'_{\mathtt{I}}} = (g^{q_1})^{x_i x'_{\mathtt{I}}} = h_2^{x_i x'_{\mathtt{I}}} \in \mathbb{G}_2,\end{aligned}$$

$$\text{and } \forall (a_i, a_j) \in \mathbb{A}^2, e(\psi(a_i), \psi(a_j)) \in \mathbb{G}_{T_2}$$

### 4.2 T-MATCH Overview

**Protocol overview** Before presenting detailed description of T-MATCH, we provide an overview of T-MATCH that summarizes how the matching protocol works.

Each tag $T_i$ in T-MATCH stores a state $s_{T_i}^{k_i}$ that consists of a BGN encryption $c_{T_i}^{k_i} = \texttt{Enc}_{\mathbb{G}}(\psi(a_{T_i})) = \texttt{Enc}_{\mathbb{G}}(H(a_{T_i})^{x_{\texttt{I}}})$ of $T_i$'s attribute $a_{T_i}$ (where $H : \mathbb{Z}_N \rightarrow \mathbb{G}_T$ is a cryptographic hash function, and $x_{\texttt{I}}$ is the issuer $\texttt{I}$'s secret key), together with a keyed HMAC $\sigma_{T_i}^{k_i} = \texttt{HMAC}_K(c_{T_i}^{k_i})$, i.e., $s_{T_i}^{k_i} = (c_{T_i}^{k_i}, \sigma_{T_i}^{k_i})$. On the other hand, backend server $\texttt{S}$ stores a set $\texttt{REF}$ of $\nu$ matching references. Each matching reference $\texttt{Ref}_{(i,j)}$ corresponds to two attributes $a_i$ and $a_j$ in $\mathbb{A}$ that match and it is computed as:

$$\texttt{Ref}_{(i,j)} = f(a_i, a_j) = f(a_j, a_i) = e(\psi(a_i), \psi(a_j)) = e(H(a_i)^{x_{\texttt{I}}}, H(a_j)^{x_{\texttt{I}}})$$

When two tags $T_i$ and $T_j$ come together in the range of a reader $\texttt{R}_k$, reader $\texttt{R}_k$ reads the current states $s_{T_i}^{k_i}$ and $s_{T_j}^{k_j}$ of tags $T_i$ and $T_j$'s respectively. Reader $\texttt{R}_k$ checks first, whether the keyed $\texttt{HMAC}$ stored into tags $T_i$ and $T_j$ are valid or not. If they are, reader $\texttt{R}_k$ computes the bilinear pairing $e(c_{T_i}^{k_i}, c_{T_j}^{k_j})$.

$$\begin{aligned} C_{(i,j)} &= e(c_{T_i}^{k_i}, c_{T_j}^{k_j}) = e(\texttt{Enc}_{\mathbb{G}}(\psi(a_{T_i})), \texttt{Enc}_{\mathbb{G}}(\psi(a_{T_j}))) \\ &= \texttt{Enc}_{\mathbb{G}_T}(e(\psi(a_{T_i}), \psi(a_{T_j}))) \end{aligned}$$

Next, reader $\texttt{R}_k$ and backend server $\texttt{S}$ engage in a secure two party protocol for *plaintext equality test* (PET) [9] to check whether the underlying plaintext of ciphertext $C_{(i,j)}$ belongs to the set of matching references $\texttt{REF}$ of backend server $\texttt{S}$ or not. That is, reader $\texttt{R}_k$ and backend server $\texttt{S}$ check whether:

$$\exists\, \texttt{Ref}_p \in \texttt{REF},\ C_{(i,j)} = \texttt{Enc}_{\mathbb{G}_T}(\texttt{Ref}_p)$$

Now, a reader $\texttt{R}_k$ outputs $b = 1$ (i.e., $\textsc{Check}(T_i, T_j) = 1$), if the plaintext equality test outputs 1; otherwise, it outputs $b = 0$.

**Privacy and security overview** To protect the privacy of tags, a tag $T_i$ in T-MATCH stores a BGN encryption of its attribute $a_{T_i}$ and a keyed $\texttt{HMAC}$ of the encryption. In each protocol execution, the BGN encryption is re-encrypted by readers $\texttt{R}_k$ and the $\texttt{HMAC}$ is computed accordingly. Also, neither readers $\texttt{R}_k$ nor backend server $\texttt{S}$ can decrypt the BGN ciphertext stored into $T_i$, unless readers $\texttt{R}_k$ and backend server $\texttt{S}$ collude to perform a threshold BGN decryption.

Now, to protect the privacy of tags that participate in the matching protocol against readers $\texttt{R}_k$ and backend server $\texttt{S}$, we rely on a *modified* privacy preserving plaintext equality test that is run jointly by reader $\texttt{R}_k$ and backend server $\texttt{S}$. Moreover, T-MATCH uses shuffling techniques to ensure that the only information leaked at the end of the matching protocol is a bit $b$ that indicates whether the pair of tags participating in the current execution of T-MATCH match or not.

Furthermore, to prevent backend server $\texttt{S}$ from forging new matching references from the set $\texttt{REF}$, a matching reference in T-MATCH is computed as a bilinear pairing.

Finally, T-MATCH encodes attributes as a signature of issuer $\texttt{I}$ to prevent readers $\texttt{R}_k$ from creating new tags, and it relies on a keyed $\texttt{HMAC}$ to impede adversaries from tampering with tags' content without being detected.

### 4.3 Protocol description

We now describe in more details how T-MATCH performs tag matching.

**System setup** A trusted third party (TTP) outputs a matching pair of BGN public key $\texttt{pk} = (N, \mathbb{G}, \mathbb{G}_T, e, g, h_1)$ and secret key $\texttt{sk} = q_1$, a cryptographic hash function $H : \mathbb{Z}_N \rightarrow \mathbb{G}_T$, a secret key $x_{\texttt{I}} = q_1 x'_{\texttt{I}}$ where $x'_{\texttt{I}}$ is selected randomly in $\mathbb{Z}_N^*$, and an HMAC key $K$. The TTP selects randomly a secret share $\alpha_1 \in \mathbb{Z}_N$. Then, the TTP sets the second secret share to $\alpha_2 = \texttt{sk} - \alpha_1 = q_1 - \alpha_1 \bmod N$.

Next, the TTP computes the set $\texttt{REF}$ of matching references. On input of attribute $a_i \in \mathbb{A}$, TTP computes $\psi(a_i) = H(a_i)^{x_{\texttt{I}}} \in \mathbb{G}_2$. If two attributes $a_i$ and $a_j$ match, then the TTP computes the corresponding matching reference $\texttt{Ref}_{(i,j)} = e(\psi(a_i), \psi(a_j)) = e(\psi(a_j), \psi(a_i)) \in \mathbb{G}_{T_2}$.

Finally, the TTP supplies

– each reader $\mathtt{R}_k$ with its share $\alpha_1$ of secret key $\mathtt{sk}$ and with the $\mathtt{HMAC}$ key $K$;
– backend server $\mathtt{S}$ with its share $\alpha_2$ of secret key $\mathtt{sk}$ and with the set of matching references $\mathtt{REF}$;
– issuer $\mathtt{I}$ with hash function $H$, secret key $x_{\mathtt{I}} = q_1 x'_{\mathtt{I}}$ and the $\mathtt{HMAC}$ key $K$.

**Tag initialization (Issuer $\mathtt{I}$ → Tag $T_i$)** For each new tag $T_i$, issuer $\mathtt{I}$ computes $\psi(a_{T_i}) = H(a_{T_i})^{x_{\mathtt{I}}}$, such that $a_{T_i}$ is the attribute associated with the chemical in the container that $T_i$ will label. Then, using the BGN public key $\mathtt{pk}$, issuer $\mathtt{I}$ picks a random number $r_i^0$ and computes a ciphertext $c_{T_i}^0 = \mathtt{Enc}_{\mathbb{G}}(\psi(a_{T_i})) = \psi(a_{T_i}) h_1^{r_i^0}$. Finally, issuer $I$ computes $\sigma_i^0 = \mathtt{HMAC}_K(c_i^0)$ and stores into tag $T_i$ the state $s_{T_i}^0 = (c_i^0, \sigma_i^0)$.

**Tag matching** We break down the tag matching protocol into three operations that describe, **first**, the interaction between tags $T_i$, $T_j$ and reader $\mathtt{R}_k$, **second**, the interaction between reader $\mathtt{R}_k$ and backend server $\mathtt{S}$, and **third** the computation of the output of the CHECK function by reader $\mathtt{R}_k$:

– ***Tag*** $T_i$ ↔ ***Reader*** $\mathtt{R}_k$ ↔ ***Tag*** $T_j$**:** Assume there are two tags $T_i$ and $T_j$ in the range of reader $R$. Tags $T_i$ and $T_j$ store states $s_{T_i}^{k_i} = (c_{T_i}^{k_i}, \sigma_{T_i}^{k_i})$ and $s_{T_j}^{k_j} = (c_{T_j}^{k_j}, \sigma_{T_j}^{k_j})$ respectively.
Reader $\mathtt{R}_k$ first reads out the tags $T_i$ and $T_j$ and checks whether $\sigma_{T_i}^{k_i} = \mathtt{HMAC}_K(c_{T_i}^{k_i})$ and $\sigma_{T_j}^{k_j} = \mathtt{HMAC}_K(c_{T_j}^{k_j})$ or not. If not, reader $\mathtt{R}_k$ updates the states of tags $T_i$ and $T_j$ and aborts the protocol. Otherwise, it updates the states of tags $T_i$ and $T_j$ and continues the execution of the protocol.
Now to update the state of tag $T_i$ participating in the protocol, reader $\mathtt{R}_k$ proceeds as follows.
- If $\sigma_{T_i}^{k_i} = \mathtt{HMAC}_K(c_{T_i}^{k_i})$, then reader $\mathtt{R}_k$ picks a random numbers $r'_i$ and re-encrypts the ciphertexts $c_{T_i}^{k_i}$ to obtain new BGN ciphertext $c_{T_i}^{k_i+1} = c_{T_i}^{k_i} h_1^{r'_i}$. Then, it computes $\sigma_{T_i}^{k_i+1} = \mathtt{HMAC}_K(c_{T_i}^{k_i+1})$. Finally, reader $\mathtt{R}_k$ writes the new state $s_{T_i}^{k_i+1} = (c_{T_i}^{k_i+1}, \sigma_{T_i}^{k_i+1})$ into tag $T_i$.
- If $\sigma_{T_i}^{k_i} \neq \mathtt{HMAC}_K(c_{T_i}^{k_i})$, then reader $\mathtt{R}_k$ picks two random strings $(st_1, st_2)$ and stores them into tag $T_i$.

– ***Reader*** $\mathtt{R}_k$ → ***Backend server*** $\mathtt{S}$**:** Reader $\mathtt{R}_k$ then computes the BGN ciphertext $C_{(i,j)} = e(c_{T_i}^{k_i}, c_{T_j}^{k_j}) \in \mathbb{G}_T$. Without loss of generality, we assume that $c_{T_i}^{k_i} = \mathtt{Enc}_{\mathbb{G}}(\psi(a_{T_i})) = \psi(a_{T_i}) h_1^{r_i}$ and $c_{T_j}^{k_j} = \mathtt{Enc}_{\mathbb{G}}(\psi(a_{T_j})) = \psi(a_{T_j}) h_1^{r_j}$. By bilinearity of $e$:

$$\begin{aligned} C_{(i,j)} &= e(c_{T_i}^{k_i}, c_{T_j}^{k_j}) = e(\psi(a_{T_i}) h_1^{r_i}, \psi(a_{T_j}) h_1^{r_j}) \\ &= e(\psi(a_{T_i}), \psi(a_{T_j}) h_1^{r_j}) \cdot e(h_1^{r_i}, \psi(a_{T_j}) h_1^{r_j}) \\ &= e(\psi(a_{T_i}), \psi(a_{T_j})) \cdot e(\psi(a_{T_i}), h_1^{r_j}) \cdot e(h_1^{r_i}, \psi(a_{T_j})) \cdot e(h_1^{r_i}, h_1^{r_j}) \end{aligned}$$

We recall that:
- $h_1 = u^{q_2}$ where $u$ is a generator of $\mathbb{G}$, and that there exist $x \in \mathbb{Z}_N$ such that $h_1 = g^x$;
- $\psi(a_{T_i})$ and $\psi(a_{T_j})$ are elements of $\mathbb{G}_2$ and that $h_2 = g^{q_1}$ is generator of $\mathbb{G}_2$. As a result, there exist $x_i,\ x_j \in \mathbb{Z}_N$ such that $\psi(a_{T_i}) = h_2^{x_i} = g^{q_1 x_i}$ and $\psi(a_{T_j}) = h_2^{x_j} = g^{q_1 x_j}$.

$$\begin{aligned} C_{(i,j)} &= e(\psi(a_{T_i}), \psi(a_{T_j})) \cdot e(g^{q_1 x_i}, u^{q_2 r_j}) \cdot e(u^{q_2 r_i}, g^{q_1 x_j}) \cdot e(g^{x r_i}, h_1^{r_j}) \\ &= e(\psi(a_{T_i}), \psi(a_{T_j})) \cdot e(g^{x_i}, u^{r_j})^{q_1 q_2} \cdot e(u^{r_i}, g^{x_j})^{q_1 q_2} \cdot e(g, h_1)^{x r_i r_j} \\ &= e(\psi(a_{T_i}), \psi(a_{T_j})) \cdot \underbrace{e(g^{x_i}, u^{r_j})^N}_{1} \cdot \underbrace{e(u^{r_i}, g^{x_j})^N}_{1} \cdot e(g, h_1)^{x r_i r_j} \\ &= e(\psi(a_{T_i}), \psi(a_{T_j})) \cdot e(g, h_1)^R \end{aligned}$$

where $R = x r_i r_j$ is uniformly distributed in $\mathbb{Z}_N$, thus:

$$C_{(i,j)} = \mathtt{Enc}_{\mathbb{G}_T}(e(\psi(a_{T_i}), \psi(a_{T_j})))$$

This directly follows from the homomorphic property of BGN as illustrated in Section 4.1.
Reader $\mathtt{R}_k$ then sends ciphertext $C_{(i,j)}$ to backend server $\mathtt{S}$.

– ***Backend server*** $\mathtt{S}$ → ***Reader*** $\mathtt{R}_k$**:** Without loss of generality, we assume that $\mathtt{REF} = \{\mathtt{Ref}_1, \mathtt{Ref}_2, ..., \mathtt{Ref}_\nu\}$, and that for all $\mathtt{Ref}_p \in \mathtt{REF}$, there exist $a_i$ and $a_j$ in $\mathbb{A}$, such that $\mathtt{Ref}_p = e(\psi(a_i), \psi(a_j))$.
Upon receiving ciphertext $C_{(i,j)}$ from reader $\mathtt{R}_k$, backend server $\mathtt{S}$ proceeds as follows:

- It picks $\nu$ random numbers $R_p \in \mathbb{Z}_N^*$, and computes $\nu$ ciphertexts $C_p = \left(\frac{C_{(i,j)}}{\texttt{Ref}_p}\right)^{R_p}$, for all $p$ in $\{1, 2, ..., \nu\}$.
- On input of its secret share $\alpha_2$ and ciphertexts $C_p$, backend server S computes $M'_p = (M_{1,p}, M_{2,p}) = (C_p, C_p^{\alpha_2})$. Next, backend server S shuffles $M'_p$.
  We note that by shuffling messages $M'_p$, T-MATCH prevents semi-honest readers $\texttt{R}_k$ from telling whether two pairs of matching tags satisfy the same matching reference or not.
- Finally, backend server S sends $M'_p$ to reader $\texttt{R}_k$.

– ***The output of the*** CHECK ***function*****:** When receiving $M'_p$ from backend server S, reader $\texttt{R}_k$ uses its secret share $\alpha_1$ and computes:

$$M_p = {M_{1,p}}^{\alpha_1} \cdot M_{2,p} = C_p^{\alpha_1} \cdot C_p^{\alpha_2} = C_p^{\alpha_1+\alpha_2} = C_p^{q_1} = \left(\left(\frac{C_{(i,j)}}{\texttt{Ref}_p}\right)^{R_p}\right)^{q_1}$$
$$= \left(\left(\frac{e(\psi(a_{T_i}), \psi(a_{T_j})) \cdot e(g, h_1)^R}{\texttt{Ref}_p}\right)^{R_p}\right)^{q_1}$$
$$= \left(\frac{e(\psi(a_{T_i}), \psi(a_{T_j}))}{\texttt{Ref}_p}\right)^{q_1 R_p} \cdot e(g, h_1)^{q_1 R R_p}$$
$$= \left(\frac{e(\psi(a_{T_i}), \psi(a_{T_j}))}{\texttt{Ref}_p}\right)^{q_1 R_p}$$

Note that if $T_i$ and $T_j$ match then there exists a matching reference $\texttt{Ref}_p \in \texttt{REF}$ such that: $e(\psi(a_{T_i}), \psi(a_{T_j})) = \texttt{Ref}_p$ and thus,

$$M_p = \left(\frac{e(\psi(a_{T_i}), \psi(a_{T_j}))}{\texttt{Ref}_p}\right)^{q_1 R_p} = 1$$

Consequently, if there exists $p \in \{1, 2, ..., \nu\}$ such that $M_p = 1$, then reader $\texttt{R}_k$ outputs $b = 1$ meaning that $T_i$ and $T_j$ match. Otherwise, $\texttt{R}_k$ outputs $b = 0$, i.e., $T_i$ and $T_j$ do not match.

# 5 Security and privacy analysis

# 6 Security analysis

In the following section, we state the security theorems of T-MATCH.

We recall that backend server S and reader $\texttt{R}_k$ are semi-honest, and that issuer I is honest.

## 6.1 Completeness

**Theorem 1.** T-MATCH *is complete.*

*Proof (Sketch).* If two tags $T_i$ and $T_j$ store attributes $a_{T_i}$ and $a_{T_j}$ that match, then there is $\texttt{ref} \in \texttt{REF}$, such that $\texttt{ref} = e(\psi(a_{T_i}), \psi(a_{T_j}))$. Therefore, one of the $\nu$ messages $M_p$ computed by reader $\texttt{R}_k$ will be equal to 1. Consequently, reader $\texttt{R}_k$ will output $b = 1$, i.e., the output of CHECK$(T_i, T_j)$ is 1. □

## 6.2 Soundness

To prove the soundness of T-MATCH, we use the following lemma

**Lemma 1.** *If* $R_p \in \mathbb{Z}_N^*$*, then* $M_p = 1 \Leftrightarrow e(\psi(a_{T_i}), \psi(a_{T_j})) = \texttt{Ref}_p$

*Proof.* We recall that for all $a_i \in \mathbb{A}$, $\psi(a_i) \in \mathbb{G}_2$. By construction, $h_2 = g^{q_1}$ is a generator of $\mathbb{G}_2$. As a result, for all $a_i \in \mathbb{A}, \exists\, x_i \in \mathbb{Z}_{q_2}$ such that $\psi(a_i) = h_2^{x_i} = g^{q_1 x_i}$. Consequently, there exist $x_i, x_j, x_p \in \mathbb{Z}_{q_2}$ such that: $e(\psi(a_{T_i}), \psi(a_{T_j})) = e(g, g)^{q_1^2 x_i x_j}$ and $\texttt{Ref}_p = e(g, g)^{q_1^2 x_p}$. Thus,

$$M_p = \left(\frac{e(\psi(a_{T_i}), \psi(a_{T_j}))}{\texttt{Ref}_p}\right)^{q_1 R_p} = e(g, g)^{q_1^3 x R_p}$$

Where $x = x_i x_j - x_p \bmod N$.

If $e(g,g)^{q_1^3 x R_p} = 1$, then this implies that $q_1^3 x R_p = 0 \bmod N$. Since $R_p \in \mathbb{Z}_N^*$, then $q_1^3 x = 0 \bmod N$ and therewith $x = x_i x_j - x_p = 0 \bmod q_2$.

We conclude that $x_i x_j = x_p \bmod q_2$ and $q_1^2 x_i x_j = q_1^2 x_p \bmod N$. Consequently, $e(\psi(a_{T_i}), \psi(a_{T_j})) = \mathtt{Ref}_p$.

**Theorem 2.** T-MATCH *is sound under the security of* HMAC *and the security of the hash function* $H$.

*Proof (Sketch).* If there is an adversary $\mathcal{A}$ who breaks the soundness property of T-MATCH, then this implies that adversary $\mathcal{A}$ is able to provide reader R with a pair of tags $\mathtt{T}_0$ and $\mathtt{T}_1$ such that:

**i.)** Tag $\mathtt{T}_0$ (respectively $\mathtt{T}_1$) stores a state $s_{\mathtt{T}_0} = (c_{\mathtt{T}_0}, \mathtt{HMAC}_K(c_{\mathtt{T}_0}))$ (respectively $s_{\mathtt{T}_1} = (c_{\mathtt{T}_1}, \mathtt{HMAC}_K(c_{\mathtt{T}_1}))$);
**ii.)** CHECK$(\mathtt{T}_0, \mathtt{T}_1) = 1$, i.e., there exists a matching reference $\mathtt{Ref}_{(p,q)} = e(\psi(a_p), \psi(a_q))$ that matches the pair of tags $\mathtt{T}_0$ and $\mathtt{T}_1$.
**iii.)** and finally, $\{\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_0}), \mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_1})\} \neq \{\psi(a_p), \psi(a_q)\}$.

There are two cases to consider, depending on whether $\mathtt{T}_0$ and $\mathtt{T}_1$ encode valid attributes or not.

– **Case 1:** $\mathtt{T}_0$ and $\mathtt{T}_1$ encode valid attributes, i.e., there exist $a_i, a_j \in \mathbb{A}$ such that $\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_0}) = \psi(a_i)$ and $\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_1}) = \psi(a_j)$. Breaking the soundness property of T-MATCH implies that there exist $\{a_i, a_j\} \neq \{a_p, a_q\} \subset \mathbb{A}$ such that $\mathtt{ref}_{(p,q)} = e(\psi(a_p), \psi(a_q)) = e(\psi(a_i), \psi(a_j))$ using Lemma 1.
  - Let $E$ denote the event that for all $\{a_i, a_j\} \neq \{a_p, a_q\} \subset \mathbb{A}$, $e(\psi(a_i), \psi(a_j)) \neq e(\psi(a_p), \psi(a_q))$.
  - Let $\bar{E}$ denote the event that there exists $\{a_i, a_j\} \neq \{a_p, a_q\} \subset \mathbb{A}$, such that $e(\psi(a_i), \psi(a_j)) = e(\psi(a_p), \psi(a_q))$.

  Assuming that $H : \mathbb{Z}_N \rightarrow \mathbb{G}$ is a cryptographic hash function implies that for all $a_i \in \mathbb{A}$, $\psi(a_i) = H(a_i)^{x_1} = H(a_i)^{q_1 x_1'}$ is uniformly distributed in $\mathbb{G}_2$. That is, for all $a_i \in \mathbb{A}$ there exists $x_i$ uniformly distributed in $\mathbb{Z}_{q_2}$ such that $\psi(a_i) = h_2^{x_i}$ ( we recall that $h_2$ is a random generator of $\mathbb{G}_2$ and that the order of $\mathbb{G}_2$ is $q_2$). Along these lines, for any pair of attributes $(a_i, a_j) \in \mathbb{A}$, $e(\psi(a_i), \psi(a_j)) = e(h_2, h_2)^{x_i x_j}$ is distributed uniformly in the subgroup $\mathbb{G}_{T_2}$.
  - Let $\mathtt{P}_{\mathbb{A}}$ denote the set of all possible pairs in $\mathbb{A}$ and $L$ denote the number of these pairs, i.e., $L = |\mathtt{P}_{\mathbb{A}}| = \frac{l(l-1)}{2}$. Without loss of generality, $\mathtt{P}_{\mathbb{A}} = \{\mathtt{p}_1, \mathtt{p}_2, ..., \mathtt{p}_L\}$.
  - Let $E_i$ denote the event that pair $\mathtt{p}_i$ in $\mathtt{P}_{\mathbb{A}}$ does not have the same matching reference as pairs $\{\mathtt{p}_1, \mathtt{p}_2, ..., \mathtt{p}_{i-1}\}$.

  We recall that $|q_2| = 512$ bits where $q_2$ is the order of $\mathbb{G}_{T_2}$, and that typically $|l| \leq 10$ bits. Now, the probability of event $E$ is:

$$Pr(E) = \prod_{i=1}^{L} \Pr(E_i) = 1\left(1 - \frac{1}{q_2}\right)\left(1 - \frac{2}{q_2}\right)...\left(1 - \frac{L-1}{q_2}\right)$$
$$\geq \left(1 - \frac{L-1}{q_2}\right)^L \simeq \left(1 - \frac{2^{2|l|}}{2^{|q_2|}}\right)^L \simeq \left(1 - L\frac{2^{2|l|}}{2^{|q_2|}}\right) \simeq \left(1 - \frac{2^{4|l|}}{2^{|q_2|}}\right)$$

  Consequently, $Pr(\bar{E}) = 1 - Pr(E) \simeq 2^{4|l| - |q_2|}$. Thus, the probability that event $\bar{E}$ occurs is negligible. We conclude that given the security of the hash function $H$, the probability that an adversary $\mathcal{A}$ breaks the soundness property when tags $\mathtt{T}_0$ and $\mathtt{T}_1$ encode valid attributes is negligible.
– **Case 2:** $\mathtt{T}_0$ or $\mathtt{T}_1$ does not encode valid attributes, i.e., for all $a_p \in \mathbb{A}$, $\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_0}) \neq \psi(a_p)$ or $\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_1}) \neq \psi(a_p)$.
  Without loss of generality, we assume that for all $a_p \in \mathbb{A}$, $\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_0}) \neq \psi(a_p)$.
  Now, if for all $a_p \in \mathbb{A}$ $\mathtt{Dec}_{\mathbb{G}}(c_{\mathtt{T}_0}) \neq \psi(a_p)$, then this implies that tag $\mathtt{T}_0$ was not issued by issuer I. Consequently, $\mathtt{T}_0$'s state $s_{\mathtt{T}_0}^k = (c_{\mathtt{T}_0}^k, \sigma_{\mathtt{T}_0}^k)$ was necessarily computed by adversary $\mathcal{A}$. As a result, adversary $\mathcal{A}$ is able to compute the HMAC of $c_{\mathtt{T}_0}$ without the secret key $K$. This leads to a contradiction under the security of HMAC.
  We conclude that given the security of HMAC, an adversary $\mathcal{A}$ cannot break the soundness of T-MATCH when tag $\mathtt{T}_0$ or tag $\mathtt{T}_1$ does not encode valid attributes. □

## 7 Privacy

In the this section, we present T-MATCH's privacy theorems.

### 7.1 Privacy of readers $\mathtt{R}_k$ and backend server $\mathtt{S}$

**Theorem 3.** T-MATCH *preserves the privacy of readers* $\mathtt{R}_k$ *and backend server* $\mathtt{S}$ *in the semi-honest model under the subgroup decision assumption in* $\mathbb{G}_T$.

*Proof (sketch).* We need to show how to transform any admissible pair $(\mathcal{A}_1, \mathcal{A}_2)$ of adversaries against T-MATCH in the real model, into an admissible pair $(\mathcal{B}_1, \mathcal{B}_2)$ of adversaries in the ideal model.
**– Backend server $\mathtt{S}$ is honest.** First, we start with the case of an honest backend server $\mathtt{S}$. In this case, $\mathcal{B}_2$ is the honest backend server $\mathtt{S}$ in the ideal model, and we transform the adversary $\mathcal{A}_1$ (semi-honest reader $\mathtt{R}_k$) against $\mathtt{S}$ in the real model into an adversary $\mathcal{B}_1$ (semi-honest reader $\mathtt{R}_k$) against $\mathtt{S}$ in the ideal model.

Adversary $\mathcal{B}_1$ will execute $\mathcal{A}_1$ locally, obtaining therefore the messages that $\mathcal{A}_1$ would have sent in a real execution of T-MATCH, and providing $\mathcal{A}_1$ with the messages that he expects to receive from backend server $\mathtt{S}$.

- $\mathcal{A}_1$ reads the states $s_{T_i}^{k_i}$ and $s_{T_j}^{k_j}$ stored into tags $T_i$ and $T_j$ respectively, and computes the bilinear pairing $C_{(i,j)}$ of the ciphertexts stored into $T_i$ and $T_j$. Then, $\mathcal{A}_1$ sends $C_{(i,j)}$ to $\mathcal{B}_1$ who simulates backend server $\mathtt{S}$.
- $\mathcal{B}_1$ sends the ciphertexts stored into tags $T_i$ and $T_j$ and the secret share $\alpha_1$ to the trusted third party.
- $\mathcal{B}_1$ receives a bit $b$ from the TTP which is the output of the CHECK function.
- To simulate backend server $\mathtt{S}$ to adversary $\mathcal{A}_1$, $\mathcal{B}_1$ computes $\nu$ messages $M'_p$ such that:
  1. If $b = 1$: $\mathcal{B}_1$ picks $\nu - 1$ pairs of random numbers $(x_p, R_p)$ in $\mathbb{Z}_N^*$, and computes: $M'_p = (M_{1,p}, M_{2,p}) = (e(h_1, g)^{R_p}, e(g, g)^{x_p} e(h_1, g)^{-\alpha_1 R_p})$, where $\alpha_1$ is the secret share of $\mathcal{A}_1$ (note that $M_p = M_{1,p}^{\alpha} M_{2,p} = e(g, g)^{x_p}$ is randomly distributed in $\mathbb{G}_T$). Next, $\mathcal{B}_1$ selects a random number $R_\nu \in \mathbb{Z}_N$ and computes: $M'_\nu = (M_{1,\nu}, M_{2,\nu}) = (e(h_1, g)^{R_\nu}, e(h_1, g)^{-\alpha_1 R_\nu})$.
  2. If $b = 0$: $\mathcal{B}_1$ picks $\nu$ pairs of random numbers $(x_p, R_p)$ in $\mathbb{Z}_N^*$, and computes: $M'_p = (M_{1,p}, M_{2,p}) = (e(h_1, g)^{R_p}, e(g, g)^{x_p} e(h_1, g)^{-\alpha_1 R_p})$.
- Finally, $\mathcal{B}_1$ shuffles $M'_p$ and sends them to adversary $\mathcal{A}_1$.

We show that the distribution of messages $M'_p$ sent to $\mathcal{A}_1$ when $\mathcal{B}_1$ is simulating backend server $\mathtt{S}$ is computationally indistinguishable from the distribution of messages $M'_p$ that $\mathcal{A}_1$ actually receives from backend server $\mathtt{S}$.

In fact, when adversary $\mathcal{A}_1$ runs T-MATCH in the real model, he expects to receive $\nu$ messages $M'_p$ distributed as described below:

- Tags $T_i$ and $T_j$ match: there exists a message $M'_q = (M_{1,q}, M_{2,q})$ such that $M_q = (M_{1,q})^{\alpha_1} \cdot M_{2,q} = 1$, and for all $M'_q \neq M'_p$, the product $M_p = (M_{1,p})^{\alpha_1} \cdot M_{2,p}$ is randomly distributed in $\mathbb{G}_{T_2}$.
- Tags $T_i$ and $T_j$ do not match: for all $M'_p = (M_{1,p}, M_{2,p})$, the product $M_p = (M_{1,p})^{\alpha_1} \cdot M_{2,p}$ is randomly distributed in $\mathbb{G}_{T_2}$.

Note that the resulting product $M_p = (M_{1,p})^{\alpha_1} \cdot M_{2,p}$ from the message $M'_p = (M_{1,p}, M_{2,p})$ sent by adversary $\mathcal{B}_1$ during his simulation of backend server $\mathtt{S}$ is distributed in $\mathbb{G}_T$ and not in $\mathbb{G}_{T_2}$. However, this cannot be detected by $\mathcal{A}_1$. Otherwise, this implies that $\mathcal{A}_1$ is able to tell whether an element of $\mathbb{G}_T$ is an element of the subgroup $\mathbb{G}_{T_2}$ or not, and this leads to a contradiction under the subgroup decision assumption.

Thus, $\mathcal{B}_1$ successfully simulates backend server $\mathtt{S}$ to adversary $\mathcal{A}_1$, and the distribution $\mathtt{Real}_{\bar{\mathcal{A}}}$ is computationally indistinguishable from the distribution $\mathtt{Ideal}_{\bar{\mathcal{B}}}$ when backend server $\mathtt{S}$ is honest.
**– Reader $R_k$ is honest**: In this case, $\mathcal{B}_1$ is determined and he is the honest reader $\mathtt{R}_k$. We transform next an adversary $\mathcal{A}_2$ (semi-honest backend server $\mathtt{S}$) against reader $\mathtt{R}_k$ in the real model into an adversary $\mathcal{B}_2$ (semi-honest backend server $\mathtt{S}$) against reader $\mathtt{R}_k$ in the ideal model as follows.

- $\mathcal{B}_2$ first eavesdrops on reader $\mathtt{R}_k$ to get the states of tags $T_i$ and $T_j$ participating in the matching protocol. Notice that such an attack cannot be prevented as the channel between tags and reader $\mathtt{R}_k$ is not secure in the ideal model.
- $\mathcal{B}_2$ simulates reader $\mathtt{R}_k$ for adversary $\mathcal{A}_2$ in the real model, computes the bilinear pairing $C_{(i,j)}$ of ciphertexts stored into tags $T_i$ and $T_j$. Then, $\mathcal{B}_2$ sends $C_{(i,j)}$ to adversary $\mathcal{A}_2$.
- $\mathcal{B}_2$ sends the set of matching references $\mathtt{REF}$, together with the secret share $\alpha_2$ to the TTP.
- The TTP returns a bit $b$ to reader $\mathtt{R}_k$ in the ideal model. If $b = 1$, then reader $\mathtt{R}_k$ raises an alarm in the ideal model, and so does $\mathcal{B}_2$ in the real model when simulating reader $\mathtt{R}_k$. Otherwise, $\mathcal{B}_2$ does nothing.

Note that when provided with the states of tags $T_i$ and $T_j$, and the output of the CHECK function, adversary $\mathcal{B}_2$ can successfully simulate reader $\mathtt{R}_k$ to adversary $\mathcal{A}_2$ in the real model. Hence, the distributions $\mathtt{Real}_{\bar{\mathcal{A}}}$ and $\mathtt{Ideal}_{\bar{\mathcal{B}}}$ are indistinguishable when reader $\mathtt{R}_k$ is honest.

Accordingly, T-MATCH ensures the privacy of readers $\mathtt{R}_k$ and backend server S in the semi-honest model. □

### 7.2 Tag privacy

**Theorem 4.** T-MATCH *ensures tag unlinkability under the subgroup decision assumption in* $\mathbb{G}$ *and* $\mathbb{G}_T$*, and the security of* HMAC.

*Proof.* Assume there is an adversary $\mathcal{A}$ who breaks the tag unlinkability of T-MATCH with a non-negligible advantage $\epsilon$. We show that we can build an adversary $\mathcal{B}$ who uses $\mathcal{A}$ as a subroutine and breaks the semantic security of BGN under re-encryption with a non-negligible advantage $\epsilon'$.

Let $\mathcal{O}_{\mathtt{re-encryption}}$ be the oracle that when queried with two ciphertexts $c_0$ and $c_1$ in $\mathbb{G}$, flips a random coin $b \in \{0,1\}$, re-encrypts $c_b$ using BGN and public key pk, and returns the resulting ciphertext $c'_b$.

As this re-encryption is based on BGN, the semantic security property of the BGN encryption can be extended to semantic security under re-encryption [7]. Let $\mathcal{B}$ be an adversary that selects two ciphertexts $c_0$, $c_1$ and provides oracle $\mathcal{O}_{\mathtt{re-encryption}}$ with $c_0$ and $c_1$ encrypted under public key pk. $\mathcal{O}_{\mathtt{re-encryption}}$ randomly chooses $b$, re-encrypts $c_b$ to $c'_b$ with public key pk, and returns $c'_b$ to $\mathcal{B}$.

The semantic security of BGN under re-encryption implies that guessing the value of $b$ is as difficult for $\mathcal{B}$ as the subgroup decision problem.

To break the semantic security of BGN under re-encryption, $\mathcal{B}$ proceeds as follows:

- Adversary $\mathcal{B}$ creates a T-MATCH system with $l$ attributes $\mathbb{A} = \{a_1, a_2, ..., a_l\}$, issuer I, $\eta$ readers $\mathtt{R}_k$ and backend server S.
  $\mathcal{B}$ selects a random HMAC key $K$, random secret key $x_{\mathtt{I}}$, random shares $\alpha_1$ and $(-\alpha_1)$, and a hash function $H : \mathbb{Z}_N \rightarrow \mathbb{G}$. Next, he computes the matching references REF that he is going to use to compute the output of the CHECK function.
  Then, he provides issuer I with secret keys $K$, $x_{\mathtt{I}}$ and the hash function $H$, readers $\mathtt{R}_k$ with secret key $K$ and secret share $\alpha_1$, and backend server S with secret share $(-\alpha_1)$ and the set of matching references REF.
  Finally, he simulates issuer I and initializes $n$ tags using as input $\mathbb{A}$, public key pk from the re-encryption oracle $\mathcal{O}_{\mathtt{re-encryption}}$, hash function $H$, HMAC key $K$ and secret key $x_{\mathtt{I}}$.
  At the end of tag initialization phase, each tag $T_i$ stores a state $s^0_{T_i} = (c^0_{T_i}, \sigma^0_{T_i}) = (\mathtt{Enc}_{\mathbb{G}}(\psi(a_{T_i})), \mathtt{HMAC}_K(c^0_{T_i}))$ such that $a_{T_i} \in \mathbb{A}$.
- $\mathcal{B}$ initializes a database DB where he keeps an entry $E_{T_i}$ for each tag $T_i$ such that: $E_{T_i} = (a_{T_i}, c^0_{T_i}, c^1_{T_i}, ..., c^j_{T_i}, ...)$, where $c^0_{T_i}$ is the ciphertext stored into $T_i$ at initialization, and $c^j_{T_i}$ is the ciphertext stored into tag $T_i$ after the $j^{\mathtt{th}}$ interaction of tag $T_i$ with readers $\mathtt{R}_k$.
- **Learning phase:** $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathtt{CorruptR}}$ to adversary $\mathcal{A}$.
  − If $\mathcal{A}$ calls the oracle $\mathcal{O}_{\mathtt{CorruptS}}$ without making any call to $\mathcal{O}_{\mathtt{CorruptR}}$, then $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathtt{CorruptS}}$ for adversary $\mathcal{A}$. Note that if $\mathcal{A}$ corrupts both readers $R_k$ and backend server S, his attack becomes trivial.
  − $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathtt{Tags}}$ and provides $\mathcal{A}$ with tags of his choice.
  − We now show how $\mathcal{B}$ simulates the output of the CHECK function to adversary $\mathcal{A}$. Without loss of generality, we assume that adversary $\mathcal{A}$ submits two tags $T_i$ and $T_j$ for which he wants to obtain the output of the CHECK function.
  There are two cases to consider, depending on whether $\mathcal{A}$ corrupted readers $\mathtt{R}_k$ or backend server S.
  **– Case 1. [**$\mathcal{A}$ *corrupts readers* $\mathtt{R}_k$**]**: To successfully simulate backend server S for adversary $\mathcal{A}$, $\mathcal{B}$ is required to eavesdrop on all of the tags' interactions with readers in T-MATCH, including the corrupted ones.
  To start T-MATCH for a pair of tags $T_i$ and $T_j$, $\mathcal{A}$ reads the states $s^{k_i}_{T_i} = (c^{k_i}_{T_i}, \sigma^{k_i}_{T_i})$ and $s^{k_j}_{T_j} = (c^{k_j}_{T_j}, \sigma^{k_j}_{T_j})$ of tags $T_i$ and $T_j$ respectively, and writes into tags $T_i$ and $T_j$ the new states $s^{k_i+1}_{T_i} = (c^{k_i+1}_{T_i}, \sigma^{k_i+1}_{T_i})$ and $s^{k_j+1}_{T_j} = (c^{k_j+1}_{T_j}, \sigma^{k_j+1}_{T_j})$ respectively. Then, adversary $\mathcal{A}$ sends a ciphertext $C_{(i,j)} = e(c^{k_i}_{T_i}, c^{k_j}_{T_j})$ to adversary $\mathcal{B}$ who is simulating backend server S.

$\mathcal{B}$ gets the states $s_{T_i}^{k_i}, s_{T_j}^{k_j}, s_{T_i}^{k_i+1}$ and $s_{T_i}^{k_j+1}$ by eavesdropping on adversary $\mathcal{A}$. Then, $\mathcal{B}$ looks up the ciphertexts $c_{T_i}^{k_i}$ and $c_{T_j}^{k_j}$ in his database and retrieves the corresponding attributes $a_{T_i}$ and $a_{T_j}$. Then, $\mathcal{B}$ updates his database by adding the new ciphertexts $c_{T_i}^{k_i+1}$ and $c_{T_j}^{k_j+1}$ to DB accordingly. That is,

$$E_{T_i} = (a_{T_i}, c_{T_i}^0, c_{T_i}^1, ..., c_{T_i}^{k_i}, c_{T_i}^{k_i+1}) \ , \ E_{T_j} = (a_{T_j}, c_{T_j}^0, c_{T_j}^1, ..., c_{T_j}^{k_j}, c_{T_j}^{k_j+1})$$

Thanks to the security of HMAC, only readers $\mathtt{R}_k$ and adversary $\mathcal{B}$ are able to compute valid states. As a consequence, if $\mathcal{B}$ keeps track of all the ciphertexts written into tags by readers $\mathtt{R}_k$ including the ones corrupted by $\mathcal{A}$, then $\mathcal{B}$ can always evaluate the CHECK function when given a pair of tags $T_i$ and $T_j$ that store valid states $s_{T_i}^{k_i} = (c_{T_i}^{k_i}, \sigma_{T_i}^{k_i})$ and $s_{T_i}^{k_j} = (c_{T_i}^{k_j}, \sigma_{T_i}^{k_j})$ respectively. $\mathcal{B}$ is only required to look up $c_{T_i}^{k_i}$ and $c_{T_j}^{k_j}$ in his DB and to retrieve the corresponding attributes $a_{T_i}$ and $a_{T_j}$.

**Note.** The simulation of decryption presented above only works because corrupted readers $\mathtt{R}_k$ in T-MATCH are assumed to be *semi-honest*, see Section 3.

Next, $\mathcal{B}$ prepares S's answer as follows:

- $a_{T_i}$ and $a_{T_j}$ match: $\mathcal{B}$ picks $\nu - 1$ pairs of random numbers $(x_p, R_p)$ in $\mathbb{Z}_N^*$, and computes: $M'_p = (M_{1,p}, M_{2,p})$
  $= (e(g,h_1)^{R_p}, e(g,g)^{x_p} e(g,h_1)^{-\alpha_1 R_p})$. Then, $\mathcal{B}$ selects a random number $R_\nu \in \mathbb{Z}_N$ and computes: $M'_\nu = (M_{1,\nu}, M_{2,\nu}) = (e(g,h_1)^{R_\nu}, e(g,h_1)^{-\alpha_1 R_\nu})$.
- $a_{T_i}$ and $a_{T_j}$ do not match: $\mathcal{B}$ picks $\nu$ pairs of random numbers $(x_p, R_p)$ in $\mathbb{Z}_N^*$, and computes: $M'_p = (M_{1,p}, M_{2,p}) = (e(g,h_1)^{R_p}, e(g,g)^{x_p} e(g,h_1)^{-\alpha_1 R_p})$.
- Finally, $\mathcal{B}$ shuffles the messages $M'_p$ and sends them to adversary $\mathcal{A}$.

As in the proof of theorem 3, we note that when message $M'_p = (M_{1,p}, M_{2,p})$ is sent by adversary $\mathcal{B}$ during $\mathcal{B}$'s simulation of backend server S, the resulting product $M_p = (M_{1,p})^{\alpha_1} \cdot M_{2,p}$ is distributed in $\mathbb{G}_T$ and not in $\mathbb{G}_{T_2}$. However, under the subgroup decision assumption in $\mathbb{G}_T$, the distribution of messages $M'_p$ sent by $\mathcal{B}$ and the distribution of messages $M'_p$ sent by backend server S are computationally indistinguishable.

**– Case 2. [**$\mathcal{A}$ *corrupts backend server* S**]**: To simulate reader $\mathtt{R}_k$, adversary $\mathcal{B}$ does the following.

− First, when $\mathcal{B}$ wants to perform the matching protocols for a pair of tags $T_i$ and $T_j$, he reads the states $s_{T_i}^{k_i} = (c_{T_i}^{k_i}, \sigma_{T_i}^{k_i})$ and $s_{T_j}^{k_j} = (c_{T_j}^{k_j}, \sigma_{T_j}^{k_j})$ of tags $T_i$ and $T_j$ respectively, and writes into tags $T_i$ and $T_j$ the new states $s_{T_i}^{k_i+1} = (c_{T_i}^{k_i+1}, \sigma_{T_i}^{k_i+1})$ and $s_{T_j}^{k_j+1} = (c_{T_j}^{k_j+1}, \sigma_{T_j}^{k_j+1})$ respectively. Next, he looks up the ciphertexts $c_{T_i}^{k_i}$ and $c_{T_i}^{k_j}$ in his database and retrieves the corresponding attributes $a_{T_i}$ and $a_{T_j}$. Then, $\mathcal{B}$ updates his database. Finally, he computes $C_{(i,j)} = e(c_{T_i}^{k_i}, c_{T_j}^{k_j})$ and sends $C_{(i,j)}$ to adversary $\mathcal{A}$.

− Then, upon receiving the messages $M'_p$ from adversary $\mathcal{A}$, $\mathcal{B}$ checks whether $a_{T_i}$ and $a_{T_j}$ match or not. If so, $\mathcal{B}$ simulates reader $\mathtt{R}_k$ and outputs 1. Otherwise, he outputs 0.

**– Challenge phase:** $\mathcal{A}$ submits two challenge tags $\mathtt{T}_0$ and $\mathtt{T}_1$.

$\mathcal{B}$ reads the states stored into $\mathtt{T}_0$ and $\mathtt{T}_1$, and retrieves the corresponding ciphertexts $c_0$ and $c_1$ respectively. To simulate $\mathcal{O}_{\mathtt{Flip}}$, $\mathcal{B}$ queries the oracle $\mathcal{O}_{\mathtt{re-encryption}}$ with ciphertexts $c_0$ and $c_1$. $\mathcal{O}_{\mathtt{re-encryption}}$ returns a re-encryption $c'_b$ of ciphertext $c_b$, $b \in \{0,1\}$. Next, $\mathcal{B}$ computes $\sigma'_b = \mathtt{HMAC}_K(c'_b)$ and stores the state $s_{\mathtt{T}_b} = (c'_b, \sigma'_b)$ into tag $\mathtt{T}_b$. Finally, $\mathcal{B}$ returns tag $\mathtt{T}_b$ to $\mathcal{A}$.

$\mathcal{A}$ outputs his guess $b'$ for the bit $b$. Now, to break the semantic security of BGN under re-encryption, $\mathcal{B}$ outputs $b'$.

Notice that if $\mathcal{A}$ outputs $b' = 1$, then tag $\mathtt{T}_b$ corresponds to tag $\mathtt{T}_1$ and therewith, $c'_b$ is a re-encryption of $c_1$. Otherwise, tag $\mathtt{T}_b$ corresponds to tag $\mathtt{T}_0$ and as a result, $c'_b$ is a re-encryption of $c_0$.

If $\mathcal{A}$ has a non-negligible advantage $\epsilon$ in breaking T-MATCH, then $\mathcal{B}$ will have a non-negligible advantage $\epsilon' = \epsilon$ in breaking the semantic security of BGN under re-encryption. This leads to a contradiction under the subgroup assumption in $\mathbb{G}$. □

## 8 Discussion

Due to limited space, we have moved the formal proofs of security and privacy to the appendix. Here, we present only their main ideas and rationale.

**Security** To prove that T-MATCH is secure against semi-honest adversaries, we rely on the security of HMAC and the security of the hash function $H$. The security of HMAC ensures that an adversary $\mathcal{A}$ who does not have the secret key $K$ cannot create a new tag $T_i$ that does not encode a valid attribute, and that can be accepted by readers $\mathtt{R}_k$. Thus, to break the security of T-MATCH, adversary $\mathcal{A}$ has to use tags that encode valid attributes (i.e., tags that were issued by issuer $I$ and updated by readers $R_k$). Now, the security of the hash function $H$ ensures that for any pair of attributes $\{a_i, a_j\} \neq \{a_p, a_q\}$, $e(\psi(a_i), \psi(a_j)) \neq e(\psi(a_p), \psi(a_q))$. Consequently, if the CHECK function outputs $b = 1$ for a pair of tags $T_i$ and $T_j$, then this implies that tags $T_i$ and $T_j$ store attributes that match.

We have moved the formal security proof to Appendix 6.

**Privacy** T-MATCH ensures the privacy of tags $T_i$, readers $R_k$ and backend server $\mathtt{S}$. First, a tag $T_i$ in T-MATCH stores a state $s_{T_i} = (c_{T_i}, \sigma_{T_i})$, where $c_{T_i}$ is a BGN encryption of $T_i$'s attribute, while $\sigma_{T_i} = \mathtt{HMAC}_K(c_{T_i})$. The state $s_{T_i}$ is updated after each read by readers $\mathtt{R}_k$. As a result, an adversary $\mathcal{A}$ who does not monitor all of $T_i$'s interactions nor does he always observe the output of CHECK cannot link tag $T_i$ to its interactions.

Second, by using secret sharing techniques, neither readers $R_k$ nor backend server $\mathtt{S}$ can disclose the encoded attribute stored into $T_i$ unless they collude and perform a threshold decryption. Moreover, backend server $\mathtt{S}$ randomizes the ciphertexts $C_p = \frac{C_{(i,j)}}{\mathtt{Ref}_p}$, for all $\mathtt{Ref}_p \in \mathtt{REF}$, then shuffles the messages $M'_p = (C_p, C_p^{\alpha_2})$. This leads that at the end of an execution of T-MATCH, the only information that a reader $\mathtt{R}_k$ learns is the output of the CHECK function.

We refer to Appendix 7 for the formal privacy proofs.

**From semi-honest adversaries to malicious adversaries** A malicious adversary $\mathcal{A} \in \{\mathtt{R}_k, \mathtt{S}\}$ is an adversary who may act arbitrarily. Adversary $\mathcal{A}$ may **i.)** refuse to participate in the protocol when the protocol is first invoked. $\mathcal{A}$ may as well **ii.)** substitute its local input: this corresponds for instance to a reader $\mathtt{R}_k$ providing an input that does not match the states of tags it has just read, or to backend server $\mathtt{S}$ submitting a set of bogus matching references as its local input. $\mathcal{A}$ may also **iii.)** abort the protocol before sending its last message.

As established by [6], a semi-honest behavior can be enforced as long as trapdoor permutations exist. The idea is to **1.)** use commitment schemes to force each party to commit to their local inputs and to generate random numbers that are uniformly distributed. In the case of T-MATCH, each reader $\mathtt{R}_k$ commits to its secret share $\alpha_1$ and the states of tags it has read, while backend server $\mathtt{S}$ commits to its secret share $\alpha_2$, its set of matching references $\mathtt{REF}$ and the randomness it uses to compute the messages $M'_p$. Then, **2.)** to ensure that the messages exchanged between reader $\mathtt{R}_k$ and backend server $\mathtt{S}$ are protocol compliant, T-MATCH can rely on zero knowledge proofs.

Whereas the above techniques enforce semi-honest behavior and ensure readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ privacy in the malicious model, they do not enforce semi-honest behavior with respect to tags participating in the protocol, as these tags are storage only and do not feature any computation capabilities. To this effect, readers $\mathtt{R}_k$ may not update the state of tags according to the protocol or they may tamper with the state of tags. However, as discussed earlier, such attacks can be detected with human inspection. Moreover, we conjecture that as long as readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ do not collude against tags, even in the malicious model, readers $\mathtt{R}_k$ and backend server $\mathtt{S}$ will only learn the outcome of the CHECK function at the end of the execution of T-MATCH.

Furthermore, we note that observing the output of the CHECK function over multiple sessions allows any adversary to infer information about tags. However, this cannot be circumvented as it is inherent to the nature of tag matching protocols.

## 9 Evaluation

T-MATCH targets read/write only tags that do no feature any computational capabilities. A tag in T-MATCH is required to store a BGN ciphertext in $\mathbb{G}$ ($|\mathbb{G}| = 1024$ bits) and an HMAC of size 160 bits, totaling a storage of 1184 bits.

We believe that T-MATCH can be deployed using current ISO 18000-3 HF tags, such as UPM RFID HF RaceTrack tags [17] that feature up to 8 kbits of memory.

**Table 1.** Evaluation of memory and computation in T-MATCH

| | Tag | Reader $\mathtt{R}_k$ | Backend server $\mathtt{S}$ |
|---|---|---|---|
| Memory | 1184 bits | $\mathtt{pk}, K, \alpha_1$ | $\mathtt{pk},\ \alpha_2,\ \mathtt{REF}$ |
| Exponentiation in $\mathbb{G}_T$ $\lvert\mathbb{G}_T\rvert = 2048$ bits | – | $\nu$ | $2\nu$ |
| Exponentiation in $\mathbb{G}$ $\lvert\mathbb{G}\rvert = 1024$ bits | – | 2 | – |
| HMAC | – | 2 | – |
| Bilinear pairing | – | 1 | – |
| Shuffle | – | – | 1 |

In each execution of T-MATCH, reader $\mathtt{R}_k$ reads two tags $T_i$ and $T_j$ and update their states as follows. Reader $\mathtt{R}_k$ re-encrypts the BGN ciphertexts $c_{T_i}$ and $c_{T_j}$ of tags $T_i$ and $T_j$ respectively. Then, reader $\mathtt{R}_k$ computes the HMAC of the re-encrypted ciphertexts. This amounts to computing two exponentiations in $\mathbb{G}$ and two keyed hash functions.

To evaluate the CHECK function, reader $\mathtt{R}_k$ computes a ciphertext $C_{(i,j)} = e(c_{T_i}, c_{T_j}) \in \mathbb{G}_T$ such that $|\mathbb{G}_T| = 2048$ bits. Then, reader $\mathtt{R}_k$ initiates a two round protocol with backend server S by sending the ciphertext $C_{(i,j)}$.

Upon receiving ciphertext $C_{(i,j)}$, backend server S performs $2\nu$ exponentiations in $\mathbb{G}_T$ (where $\nu$ is the number of matching references in REF) and obtains $\nu$ messages $M'_p$. Next, backend server S shuffles the messages $M'_p$ and sends them to reader $\mathtt{R}_k$.

Finally, when reader $\mathtt{R}_k$ receives the messages $M'_p$, it performs $\nu$ exponentiations in $\mathbb{G}_T$ and outputs the outcome of the CHECK function.

## 10 Related work

T-MATCH shows similarities to secret handshake and secret matching protocols. Nevertheless, traditional solutions for secure and privacy preserving secret matching between two parties as proposed by Ateniese et al. [2], Balfanz et al. [3] cannot be implemented in cheap RFID tags. These solutions require the computation of bilinear pairings which cannot be performed by current RFID tags.

Boneh et al. [4] propose a protocol that allows the public evaluation of 2-DNF formula on boolean variables by relying on the BGN encryption. The protocol proposed in [4] can be slightly modified to implement tag matching. However in this case, tags are required to store $O(l)$ ciphertexts of size 1024 bits where $l$ is the number of attributes in the system – rendering such an approach unrealistic.

Another approach to evaluate the CHECK function is attribute based encryption see Goyal et al. [8], Pirretti et al. [13], Sahai and Waters [15]. The idea is to associate each attribute $a_i$ in the system with some secret component of some private key sk. When two tags $T_i$ and $T_j$ that match come together, the secret key sk can be reconstructed. The reconstruction of a correct secret key sk enables reader $\mathtt{R}_k$ to decrypt some ciphertext $C$ for which it knows the underlying plaintext $M$. The tag matching is verified by checking whether $\mathtt{Dec}(C) = M$ or not. Though, the use of attribute based encryption can allow reader $\mathtt{R}_k$ to evaluate the CHECK function by itself without a backend server S, it requires either cryptographic operations on tags or their synchronization.

## 11 Conclusion

RFID tag based matching is required by many real-world supply-chain application. Matching, however, raises new security and privacy concerns. T-MATCH tackles these challenges and provides secure and privacy preserving item matching suited for resource restricted tags unable to perform computations. T-MATCH evaluates, in a privacy preserving manner, a function CHECK that on the input of two tags $T_i$ and $T_j$ outputs a bit $b$ indicating whether $T_i$ and $T_j$ match or not. T-MATCH is provably secure and privacy preserving under standard assumptions, i.e., HMAC security and the subgroup decision assumption. Finally, designed for read/write only tags, T-MATCH requires tags to store only 150 bytes.

## Bibliography


[1] G. Ateniese, J. Camenisch, and B. de Medeiros. Untraceable RFID tags via insubvertible encryption. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 92–101, New York, NY, USA, 2005. ACM. ISBN 1-59593-226-7.

[2] G. Ateniese, J. Kirsch, and M. Blanton. Secret Handshakes with Dynamic and Fuzzy Matching. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2007.

[3] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. C. Wong. Secret Handshakes from Pairing-Based Key Agreements. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, page 180, Los Alamitos, CA, USA, 2003. IEEE Computer Society. ISBN 0-7695-1940-7.

[4] D. Boneh, E-J. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *TCC*, pages 325–341, Cambridge, MA, USA, 2005.

[5] Cobis Consortium. Collaborative business items: Chemical drums use-case, 2007. http://www.cobis-online.de/files/live.stream.wvx.

[6] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521830842.

[7] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal Re-encryption for Mixnets. In *In Proceedings of the 2004 RSA Conference, Cryptographer's track*, pages 163–178. Springer-Verlag, 2002.

[8] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5.

[9] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In Tatsuaki Okamoto, editor, *Advances in Cryptology ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer Berlin / Heidelberg, 2000. ISBN 978-3-540-41404-9.

[10] A. Juels and S.A. Weis. Defining Strong Privacy for RFID. In *PerCom Workshops*, pages 342–347, White Plains, USA, 2007. ISBN 978-0-7695-2788-8.

[11] J. Katz, A. Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, pages 146–162, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Eurocrypt '98, LNCS 1403*, pages 308–318. Springer-Verlag, 1998.

[13] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure attribute-based systems. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 99–112, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5.

[14] A.R. Sadeghi, I. Visconti, and C. Wachsmann. Anonymizer-Enabled Security and Privacy for RFID. In *8th International Conference on Cryptology And Network Security – CANS'09*, Kanazawa, Ishikawa, Japan, December 2009. Springer. ISBN 978-3-642-10432-9.

[15] A. Sahai and B. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 557–557. Springer Berlin / Heidelberg, 2005.

[16] A. Shamir. How to share a secret. *Commun. ACM*, 22:612–613, November 1979. ISSN 0001-0782.

[17] UPM RFID Technology. UPM RFID HF RaceTrack RFID Tag, 2011. `http://www.rfidtags.com/upm-rfid-racetrack-rfid-tag`.