Tracker: Security and Privacy for RFID-based Supply Chains

Erik-Oliver Blass Kaoutar Elkhiyaoui Refik Molva

EURECOM, Sophia Antipolis, France
{blass|elkhiyao|molva}@eurecom.fr

Abstract. The counterfeiting of pharmaceutics or luxury objects is a major threat to supply chains today. As different facilities of a supply chain are distributed and difficult to monitor, malicious adversaries can inject fake objects into the supply chain. This paper presents TRACKER, a protocol for object genuineness verification in RFID-based supply chains. More precisely, TRACKER allows to securely identify which (legitimate) path an object/tag has taken through a supply chain. TRACKER provides privacy: an adversary can neither learn details about an object's path, nor can it trace and link objects in the supply chain. TRACKER's security and privacy is based on an extension of polynomial signature techniques for run-time fault detection using homomorphic encryption. Contrary to related work, RFID tags in this paper are *not* required to perform any computation, but only feature a few bytes of storage such as ordinary EPC Class 1 Gen 2 tags.

1 Introduction

Supply chain management is one of the major applications of RFID tags today. The tags are physically attached to objects, therewith enabling tracking of objects on their way through the steps of a supply chain. Today, RFID-based supply chain applications range from simple barcode replacements in supermarkets to more sensitive application scenarios, where tags are used for product genuineness verification, anti-counterfeiting, anti-cloning, and replica-prevention of luxury products or pharmaceutics [9, 12, 18, 24, 26]. All these scenarios and the latter in particular raise new security and privacy challenges.

First, with respect to security, it must be verifiable whether an object has taken one of the valid paths through the supply chain, i.e., the object went through a certain valid sequence of steps in the supply chain. The goal is to allow the operator or *manager* of the supply chain to be able to check the genuineness of an object by simply scanning the object's RFID tag. The problem is, though, that supply chains are physically distributed and parties involved in a supply chain (the "steps") may reside in different locations, even in different countries. The manager does neither have full control over interconnections in between steps of the supply chain, nor full control over some of the steps itself. Also, for simple feasibility reasons, it cannot be assumed that facilities of the supply chain are permanently online or synchronized with a back-end database. Consequently, supply chains today are prone to injection of faked, counterfeit or manipulated products. For example, World Health Organization (WHO) has estimated that 10% of U.S. pharmaceutical products were already counterfeit in 2005 [6]. Today, the International Chamber of Commerce estimates that counterfeiting accounts for 5-7% of world trade, relating to $600 billion per year [11]. Hence, there is a stringent requirement for a security solution to prevent an adversary from tampering with tags in order to forge faked traces through the steps of the supply chain. Some supply chains today protect products by using additional "tamper-proof" hardware, for example the familiar holograms sticking to products. However, massive deployment of any tamper proof hardware implies additional costs. To the best of our knowledge, there is no security solution available solely based on cheap, non tamper-proof RFID tags.

The second problem regards the privacy of objects in the supply chain. Typically, the manager of the supply chain does not want to reveal any information about internal details, strategic relationships and processes within the supply chain to adversaries, e.g., competitors or customers. An adversary should not be able to trace and recognize tags and objects through subsequent steps in the supply chain and therewith learn something about the internal processes of the supply chain. Similarly, by scanning an RFID tag attached to an object, the adversary should not be able to gain any knowledge about the history of that tag and the object it is attached to.

Solutions addressing these security and privacy requirements are, however, governed by the challenges of the RFID settings: RFID tags have to be cheap for massive deployments and therefore can only afford lightweight computational capabilities. Traditional security and privacy solutions would overburden tiny tags and therefore are ineligible. Moreover, the manager of the supply chain uses a hand-held RFID reader which is typically an embedded device. Consequently, the path verification at the manager should require few cryptographic operations.

Note that security and privacy requirements for RFID-based supply chain management call for more than just privacy-preserving authentication as already extensively covered in the literature, cf., Avoine [3]. As a new requirement raised by the supply chain management, the soundness of the history kept in the tags must be assured throughout the steps of the supply chain.

This paper presents TRACKER, a protocol for secure, privacy-preserving supply chain management with RFID tags. The main idea behind TRACKER is to encode paths in a supply chain using polynomial signature techniques similar to software run-time fault detection. These polynomials will be evaluated using homomorphic encryption, thereby providing security and privacy.

TRACKER's major contributions are:

- TRACKER allows to determine the exact path that each tag[1] went through in the supply chain.
- TRACKER provides provable security: an adversary cannot create new tags or modify existing ones and fake that a tag went properly through the supply chain.
- TRACKER is privacy-preserving: only the manager of the supply chain, but no adversary, can find out a tag's path. Also, TRACKER achieves *unlinkability*. An adversary cannot link tags it observes on subsequent occasions.
- To perform path verification, the manager is required to perform $O(1)$ computations per tag, i.e., the computational complexity of path verification does neither depend on the number of tags in the supply chain n, nor on the number of valid paths ν. Memory requirements scale with $O(n+\nu)$ for the manager.
- Contrary to related work such as Ouafi and Vaudenay [20] or Li and Ding [17], TRACKER does not require tags to perform *any* computation. Instead, TRACKER relies on passive tags with limited storage, such as standard EPC Class 1 Generation 2 tags. Due to lower hardware complexity, this implies less productions costs and cheaper (or cheapest) tags in comparison to related work.
- RFID readers do not need to be permanently online or synchronized with a central data-base. In the same manner, the manager is "offline".
- TRACKER detects, but does not prevent, malicious tampering with tags' internal states by any adversary.

The rest of this paper is structured as follows: after presenting a formal model for a supply chain as used throughout this paper in Section 2, we will state the problem addressed by TRACKER and the adversary model in Section 3. This also includes the security and privacy goals within TRACKER. In sections 4 and 5, we describe TRACKER's details and formally analyze and prove TRACKER's security and privacy properties.

2 Background

We use terms and expressions similar to the ones used by Ouafi and Vaudenay [20] and Vaudenay [25].

A supply chain in this paper simply denotes series of consecutive steps that a product has to pass through. The exact meaning or semantic of such a "step" in the supply chain depends on the particular application

[1] Assuming that a tag is physically connected to an object and thereby representing it, this paper uses "tag" and "object" interchangeably.

and will not be discussed here, one could imagine a step being a warehouse or a manufacturing unit. The actual business or manufacturing process that takes place during each step of a supply chain is out of the scope of this paper. From the point of view of this paper, each step of the supply chain is equipped with an RFID reader, and when a product moves to the subsequent step of a supply chain, an interaction takes place between the product's RFID tag and the reader associated with the step. Finally, a manager wants to know whether a product went through the "correct" sequence of steps in the supply chain.

2.1 Entities

The following entities exist in TRACKER:

Tags T_i: Each tag is attached to and therewith stands for a single product or object. A tag T_i features re-writable memory representing T_i's current "state" denoted $s^j_{T_i}$. The set of all possible states is denoted with $\mathcal{S}$, $s^j_{T_i} \in \mathcal{S}$, and $|\mathcal{S}|$ is a sufficiently large security parameter of TRACKER, e.g., $|\mathcal{S}| = 2^{160}$.

Issuer I: The issuer I prepares tags for deployment. While attaching a tag T_i to a product, I writes an initial state $s^0_{T_i}$ into T_i.

Readers R_k: Representing a single step in the supply chain, a reader R_k can interact with a product's tag T_i: R_k reads out T_i's current state $s^j_{T_i}$ and writes an updated state $s^{j+1}_{T_i}$ into T_i. Here, R_k uses some function f_{R_k} to generate $s^{j+1}_{T_i}$ out of $s^j_{T_i}$, i.e., $f_{R_k}(s^j_{T_i}) = s^{j+1}_{T_i}$. Each reader is assumed to be "offline", i.e., not permanently connected to the issuer, manager, other readers, or some kind of back-end database. Only during initial system preparation, we assume that issuer I can connect to readers, e.g., to send some secrets to the reader using some secure channel.

Manager M: Eventually, a tag arrives at a special step in the supply chain called a *checkpoint*. At a checkpoint, manager M wants to check a tag's genuineness or validity. M checks whether tag T_i, and therewith the tagged object, has passed through a valid ("correct") sequence of steps in the supply chain. To do so, M simply reads out the current state $s^j_{T_i}$ of T_i. Solely based on $s^j_{T_i}$, M decides whether T_i went through a valid sequence of steps. We assume that M knows which paths in a supply chain are valid or not. As with readers, M is assumed to be offline and not synchronized with the rest of the system – besides during an initial setup.

2.2 Supply Chain

Formally, a supply chain is represented by a digraph $G = (V, E)$ consisting of vertices V and edges E.

Each vertex $v \in V$ is equivalent to one *step* in the supply chain. A vertex/step v in the supply chain is uniquely associated with a reader R_i.

Each directed edge $e \in E$, $e := \overrightarrow{v_i v_j}$, from vertex v_i to vertex v_j, expresses that v_j is a possible next step to step v_i in the supply chain. This simply means that according to the organization of the supply chain, a product might proceed to step v_j after being at step v_i. If products must not advance from step v_i to v_j, then $\overrightarrow{v_i v_j} \notin E$. Note that a supply chain can include loops and reflexive edges. Whenever a product in the supply chain proceeds from step v_i to step v_j, reader R_j interacts with the product's tag.

Issuer I is represented in G by the only vertex without incoming edges v_0.

A *path* $\mathcal{P}$ is a finite sequence of steps $\mathcal{P} = \{v_0, \ldots, v_l\}$, where $\forall i \in \{0, \ldots, l-1\} : \overrightarrow{v_i v_{i+1}} \in E$, and l is the *length* of path $\mathcal{P}$. Clearly, different paths can have different path lengths.

A *valid* path $\mathcal{P}_{\text{valid}_i}$ is a special path which manager M will eventually check products for. A valid path represents a particular legitimate sequence of steps in the supply chain that M is interested in. There may be up to ν multiple different valid paths $\{\mathcal{P}_{\text{valid}_1}, \ldots, \mathcal{P}_{\text{valid}_\nu}\}$, in a supply chain.

The last step v_l of a valid path $\mathcal{P}_{\text{valid}_i} = \{v_0, \ldots, v_l\}$ represents a *checkpoint*. After tag T_i has passed through such a checkpoint, M will check for T_i's path validity.

While manager M might not know *all* possible paths in G, we assume in the following that M knows the *valid paths*, i.e., the sequences of steps, that he is willing to accept as valid.

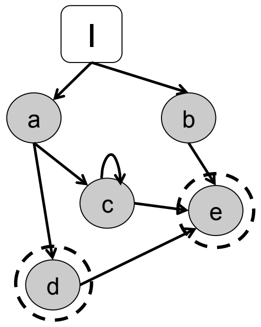

Fig. 1. Simple supply chain, checkpoints are encircled.

Figure 1 depicts a sample supply chain. Checkpoints, where manager M verifies tags/objects, are encircled. So, after their deployment at issuer I, tags can either start in steps a or b. Valid paths in Figure 1 are, for example, $\{I, a, d\}$, $\{I, a, d, e\}$ or $\{I, a, c, c, e\}$. Other sequences such as $\{I, a, e\}$ are not valid according to the supply chain.

2.3 A Tracker System

Using the above definitions, a complete TRACKER system consists of

- a supply chain $G = (V, E)$
- a set $\mathcal{T}$ of n different tags
- a set of possible states $\mathcal{S}$
- a total of η different readers, $\eta = |E|$
- issuer I and manager M
- a set of η state transition functions $f_i : \mathcal{S} \to \mathcal{S}$
- a set of ν valid paths
- a set of valid states $\mathcal{S}_{\text{valid}}$
- a database DB_{clone}, stored at manager M to protect against cloned tags (see next section)
- a function READ : $\mathcal{T} \to \mathcal{S}$ that reads out tag T_i and returns T_i's current state $s^j_{T_i}$
- a function WRITE: $\mathcal{T} \times \mathcal{S} \to \mathcal{S}$ that writes a new state $s^{j+1}_{T_i}$ into tag T_i.
- a function CHECK: $\mathcal{S} \to \begin{cases} \mathcal{P}_{\text{valid}_i}, \text{ if tag } T_i \text{ went through } \mathcal{P}_{\text{valid}_i} \\ \emptyset, \text{ if } \nexists \mathcal{P}_{\text{valid}_i} \text{ that } T_i \text{ went through} \end{cases}$

  that based on T_i's current state $s^j_{T_i}$ decides about which valid path in the supply chain tag T_i has taken.

3 Problem Statement and Adversary Model

In TRACKER, we assume that the readers in the supply chain are independent. We assume as well that a reader R_i is semi-honest ("honest-but-curious"). That is, a reader R_i at step v_i behaves correctly when it comes to the operations it has to perform on tags going through v_i. For instance, a reader R_i at step v_i that corresponds to quality control does not update the state of T unless the product attached to T satisfies the quality requirements.

Within TRACKER, we identify the following security and privacy challenges and derive a formal adversary model accordingly. Our formal definitions are direct adaptations of well-established RFID adversary models to the challenges of supply chain management. In summary, our adversary corresponds to the adversary proposed by Juels and Weis [13] and the *Non-Narrow Destructive* adversary by Vaudenay [25]

3.1 Security

The main security goal of TRACKER is to prevent an adversary from forging a tag's internal state with a valid path that was not actually taken by the tag in the supply chain. Using the components of the TRACKER system, this goal is stated as follows: **if** the verification of tag T_i's internal state $s^j_{T_i}$ by manager M using CHECK returns a valid path $\mathcal{P}_{\text{valid}_i}$, **then** T_i must have gone through the steps of $\mathcal{P}_{\text{valid}_i}$ in the supply chain.

Only the *soundness* of the CHECK function is required with respect to identification of a valid path, since the *completeness* of the CHECK function cannot always be assumed. As shown below, the adversary might write any content, for example just "garbage", into T_i at any time to spoil detection of valid paths. Even if a tag T_i has been through $\mathcal{P}_{\text{valid}_i}$ in the supply chain, the adversary might replace and invalidate the state of T_i leading to a CHECK output of "$\emptyset$".

We formalize this security property and our adversary model using game-based definitions in accordance with Juels and Weis [13].

An adversary $\mathcal{A}(\rho, r, \epsilon)$, or just $\mathcal{A}$, has access to a TRACKER system in two phases. First, in a learning phase, $\mathcal{A}$ can query an oracle $\mathcal{O}_{\text{pick}}$, cf., Algorithm 1. When queried, $\mathcal{O}_{\text{pick}}$ randomly selects a tag from all the n tags $\mathcal{T}$ in the supply chain and gives it to $\mathcal{A}$. During learning, $\mathcal{A}$ is allowed to read from and write into the tags provided by $\mathcal{O}_{\text{pick}}$. For the sake of simplicity, we assume that products and tags go through a supply chain in a clocked, synchronous way. At each "clock cycle", all tags are read and then re-written by the readers in their vicinity and then proceed to the subsequent step in the supply chain.

More precisely, the ITERATESUPPLYCHAIN command in Algorithm 1 enables $\mathcal{A}$ to iterate or "execute" the supply chain by one clock cycle, i.e., all tags advance by one step and they are read-out and re-written by readers. $\mathcal{A}$ can iterate the supply chain a total of ρ times. Now per iteration and per clock cycle, $\mathcal{A}$ gets access to a set of r arbitrary tags, read-outs their internal state, and re-writes their state with some arbitrary data. Also, $\mathcal{A}$ has access to an "oracle" like construction $\mathcal{O}_M$: queried with a tag $T_{i,j}$, $\mathcal{O}_M$ will return the output of the CHECK function.

The above definition of $\mathcal{A}$ reflects an adversary in the real world having full control over the network and knowledge about the validity of tags' states.

After the learning phase of Algorithm 1, $\mathcal{A}$ enters the (simple) challenge phase, cf., Algorithm 2.

```
for $i := 0$ to $(\rho - 1)$ do
    ITERATESUPPLYCHAIN;
    for $j := 1$ to $r$ do
        $T_{i,j} \leftarrow \mathcal{O}_{\text{pick}}$ ;
        $s^i_{T_{i,j}}$ :=READ($T_{i,j}$);
        WRITE($T_{i,j}, s^{i+1}_{T_{i,j}}$);
        CHECK($s^{i+1}_{T_{i,j}}$) $\leftarrow \mathcal{O}_M(T_{i,j})$;
    end
end
```

Algorithm 1:
Security learning phase of adversary $\mathcal{A}$

$$\left\{\begin{array}{l} T_i \leftarrow \mathcal{O}_{\text{pick}}; \\ s^j_{T_i} := \text{READ}(T_i); \\ \text{WRITE}(T_i, s^{j+1}_{T_i}); \end{array}\right\} \text{ or } \left\{\begin{array}{l} \text{CREATETAG } T_i; \\ \text{WRITE}(T_i, s^{j+1}_{T_i}) \end{array}\right\}$$

```
$\mathcal{A} \rightarrow M$: $T_i$;
$M$ evaluates CHECK on $T_i$'s state;
```

Algorithm 2:
Security challenge phase of adversary $\mathcal{A}$

$\mathcal{A}$ can either arbitrarily choose one tag $T_i \in \mathcal{T}$, read and re-write, or $\mathcal{A}$ can "create" his own tag $T_i \notin \mathcal{T}$ and write some state s'_{T_i} in it. Finally, $\mathcal{A}$ sends T_i to M. Manager M will now evaluate CHECK on T_i's state.

Definition 1 (False positives). If *M's evaluation of* CHECK *on tag T_i's state outputs one of the ν valid paths $\mathcal{P}_{\text{valid}_i} = \{v_0, \ldots, v_l\}$,* **andif** *$T_i$ has not been through the exact sequence of steps $\{v_0, \ldots, v_l\}$ in the* *supply chain,* **then** *this is called a false positive in* TRACKER*. The probability of a false positive is denoted by* $\Pr[\text{False Positive}]$.

Now, adversary $\mathcal{A}$ must not be able to generate a state corresponding to a valid path with higher probability than simple guessing:

Definition 2 (Security). TRACKER *provides security* $\Leftrightarrow$
For adversary $\mathcal{A}$*, inequality* $\Pr[\text{False Positive}] \leq \frac{|\mathcal{S}_{\text{valid}}|}{|\mathcal{S}|} + \epsilon$ *holds, where* ϵ *is negligible.*

Discussion: Cloning As we assume cheap re-writeable tags without any computational abilities, no reader authentication is possible on the tag side. Any adversary can read from and write into a tag. Trivially, an adversary might "clone" a tag. This is impossible to prevent in our setup with only re-writeable tags and offline, unsynchronized readers.

To mitigate this problem, manager M utilizes a database DB_{clone}. Initially empty, this database will contain identifiers of tags that went through a valid path of a supply chain and were checked by M. Each time that M verifies a tag's path, M will also check whether this tag's identifier is already in DB_{clone} – to check for cloning. Details about identifiers and handling of DB_{clone} will be given later in the protocol description of Section 4.

Therefore, an adversary cannot clone a tag more than once, and thus, cloning cannot be performed in a large scale. On the other hand, if the tag is attached to a luxury product, cloning is critical even if a tag is cloned only once. However, to get a malicious tag to be accepted by the manager, the adversary has to break-in the supply chain, clone a tag, inject this tag, and "overtake" the legitimate tag in the supply chain to reach the manager *before* the legitimate tag. We conjecture that this is not easy for an adversary to do.

Limitations The adversary model above does not capture an adversary hijacking tags and performing "extra" steps with tags. One might envision an adversary controlling a set of steps with readers that do not behave protocol compliant. For example, if the "extra" steps do not change the tags' state (but modify products), this will be unnoticed by the manager. We claim that these attacks, as well as physical attacks, e.g., removing one tag from one product and attaching it to another product, are out of scope.

Also, there is no notion of multiple managers in the supply chain checking tags for genuineness, but we focus on only one manager. While in the real world, multiple managers are probably more realistic, this is left for future work. Additionally, we do not target managers proving (non-) genuineness to a third party in a privacy-preserving way. Also, we focus only on detecting counterfeits, not preventing – that is, it remains unclear what happens if a counterfeit has been detected. All this is left for future research.

3.2 Privacy

An adversary in TRACKER is an active adversary who, besides being able to eavesdrop on tags' communication, can as well tamper with tags' internal states. Along these lines, we identify two notions of privacy in TRACKER: the first one is commonly known as *tag anonymity*. That is, an adversary $\mathcal{A}$ should not be able to disclose the (unique) identity of tags he reads from or writes into. The second notion of privacy that we are interested in is what we call *step privacy*: an adversary $\mathcal{A}$ should not be able to find out the steps v_i a tag went through. While $\mathcal{A}$ can eavesdrop on tags' communication and re-write tags' internal states, it should be infeasible for $\mathcal{A}$ to break tag anonymity or step privacy.

Along these lines, another notion of privacy that could be derived as well is *path privacy*: $\mathcal{A}$ should not be able to tell which path $\mathcal{P}$ a given tag T took. Note, however, that step privacy is *stronger* than path privacy. If $\mathcal{A}$ is able to disclose the path a tag T went through, then $\mathcal{A}$ automatically knows each of T's steps. So, if TRACKER preserves step privacy, then $\mathcal{A}$ cannot find out the path $\mathcal{P}$ a tag has taken.

Moreover, TRACKER should prevent $\mathcal{A}$ from binding ("linking") the data he reads to the tag storing it. This differs from tag anonymity, as the latter can be achieved, for example, through encryption. However, simple encryption cannot achieve tag unlinkability: $\mathcal{A}$ may always be able to recognize the tag through the ciphertext it stores. Thus, there is a need to regularly change the data stored on tags to prevent such a threat. In the real world, *tag unlinkability* is the property that prevents an eavesdropper from tracking, following, and distinguishing items and goods based on the data tags store.

Furthermore, $\mathcal{A}$ may as well aim at linking tags based on the steps they went through in the supply chain. Roughly speaking, *step unlinkability* should prevent an adversary $\mathcal{A}$ from telling, whether the paths that two different tags T_i and T_j took have a step in common. In practice, step unlinkability prevents an adversary $\mathcal{A}$ from binding a tag T_i to a pallet of tags in the supply chain.

In this paper, we will focus on tag unlinkability and step unlinkability for which we give formal definitions in the following section. It is sufficient to focus only on unlinkability properties, as they represent *stronger* requirements than tag anonymity and step privacy. As mentioned earlier, tag unlinkability grasps the ability of an adversary $\mathcal{A}$ to distinguish between tags based on the content they store. This notion of unlinkability is stronger than tag anonymity: if an adversary is able to undermine tag anonymity and to uniquely identify a tag, he is automatically able to distinguish tags, therewith undermining tag unlinkability. Just as well, step unlinkability ensures that it is infeasible for an adversary $\mathcal{A}$ to tell whether the paths of two tags have a step in common or not. This notion is stronger than step privacy: if $\mathcal{A}$ is able to disclose the steps any tag went through, he can always tell whether two tags have a step in common. Therefore, if TRACKER provides tag and path unlinkability, it provides as well tag anonymity and step privacy.

So in conclusion, it is sufficient to investigate unlinkability properties. These will be presented in the following section in detail.

3.3 Unlinkability

For our formal definitions of tag and path unlinkability, we assume $\mathcal{A}$ has access to the following oracles:

$\mathcal{O}_{\text{choose}}$ is an oracle that, when queried, returns a random tag T entering the supply chain.

$\mathcal{O}_{\text{select}}$ is an oracle that, when queried, returns a pair (T, S). T is a tag selected randomly from the set of tags $\mathcal{T}$ and S is the set of steps that T went through so far.

$\mathcal{O}_{\text{draw}}$ is an oracle that, when queried with a step v, returns a pair (T, S). T is a random tag that will go through v in the next supply chain iteration, and S is the set of steps that T went through so far.

$\mathcal{O}_{\text{step}}$ is an oracle that, when queried with a tag T, returns the next step that T will go through in the next supply chain iteration.

$\mathcal{O}_{\text{flip}}$ is an oracle that, when queried with two tags T_1, T_2, randomly chooses $b \in \{1, 2\}$ and returns T_b.

Tag Unlinkability: We illustrate tag unlinkability by a formal experiment similar to the experiment by Juels and Weis [13]. In this experiment, $\mathcal{A}$ has access to tags in two phases. In the learning phase, cf., Algorithm 3, $\mathcal{O}_{\text{select}}$ provides $\mathcal{A}$ with two challenge tags T_1 and T_2, and r other tags along with the steps they went through so far. $\mathcal{A}$ iterates the supply chain ρ times. At each iteration, $\mathcal{A}$ reads from and writes into tags. He is as well provided with the step that T_1 and T_2 will go through in the next supply chain iteration. This unlinkability game reflects an adversary $\mathcal{A}$ in the real world that can follow tags in the supply chain along the steps they are going through.

In the challenge phase, cf., Algorithm 4, the supply chain is iterated first. Then, $\mathcal{A}$ is provided with tag T_b, $b \in \{1, 2\}$ through oracle $\mathcal{O}_{\text{flip}}$. $\mathcal{A}$'s goal is to output the value of b. $\mathcal{O}_{\text{select}}$ provides $\mathcal{A}$ with s other tags that he can read from and write into.

Given the data stored on T_b and the result of the different readings, $\mathcal{A}$ outputs his guess for the value of $b \in \{1, 2\}$. $\mathcal{A}$ is successful, if his guess of b is correct.

```
$(T_1, S_1) \leftarrow \mathcal{O}_{\text{select}}$;
$(T_2, S_2) \leftarrow \mathcal{O}_{\text{select}}$;
for $i := 1$ to $\rho$ do
    ITERATESUPPLYCHAIN;
    $T_1 \rightarrow \mathcal{O}_{\text{step}}$;
    $v_{T_{1,(i+1)}} \leftarrow \mathcal{O}_{\text{step}}$;
    $s^i_{T_1}$ :=READ($T_1$);
    WRITE($T_1, s'^i_{T_1}$);
    $T_2 \rightarrow \mathcal{O}_{\text{step}}$;
    $v_{T_{2,(i+1)}} \leftarrow \mathcal{O}_{\text{step}}$;
    $s^i_{T_2}$ :=READ($T_2$);
    WRITE($T_2, s'^i_{T_2}$);
    for $j := 1$ to $r$ do
        $(T_{i,j}, S_{i,j}) \leftarrow \mathcal{O}_{\text{select}}$;
        $s_{T_{i,j}}$ :=READ($T_{i,j}$);
        WRITE($T_{i,j}, s'_{T_{i,j}}$);
    end
end
```

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

```
ITERATESUPPLYCHAIN;
$T_b \leftarrow \mathcal{O}_{\text{flip}}\{T_1, T_2\}$;
$s_{T_b}$ :=READ($T_b$);
for $i := 1$ to $s$ do
    $(T'_i, S'_i) \leftarrow \mathcal{O}_{\text{select}}$;
    $s_{T'_i}$ :=READ($T'_i$);
end
OUTPUT $b$;
```

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

Definition 3 (Tag Unlinkability). TRACKER *provides tag unlinkability* $\Leftrightarrow$
For adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ outputs a correct guess$) \leq \frac{1}{2} + \epsilon$ holds, where ϵ is negligible.

Discussion: Unlinkability in between reader interactions This paper targets passive tags that only feature storage capabilities and therewith cannot perform any (cryptographic) computation. Consequently, tags cannot update their state after an interaction with a reader on their own, and tags cannot perform any kind of access control. Hence, the tag state does not change in between two protocol executions, and an adversary can easily access a tag's state. Under such circumstances, it is therefore *impossible* to provide tag unlinkability against a powerful adversary who tries to link tags in between two subsequent reader interactions (cf., formal proof by Vaudenay [25]). However, we conjecture that, in a real world scenario, an adversary cannot permanently access tags or eavesdrop tags' communications, but there is *at least one* unobserved interaction between a tag and a reader. This is also in accordance with related work, such as Ateniese et al. [2], Dimitrou [8], Sadeghi et al. [22]. We implement this in our definition of adversary $\mathcal{A}$ in Algorithm 4 by iterating the supply chain before calling oracle $\mathcal{O}_{\text{flip}}$ and giving tag T_b to $\mathcal{A}$.

Step unlinkability: TRACKER should prevent an adversary $\mathcal{A}$ from being able to tell, whether the paths of two different tags T_i and T_j have a step in common.

This is formalized as follows: in the learning phase, cf., Algorithm 5, $\mathcal{A}(\rho, r, s, \epsilon)$ calls $\mathcal{O}_{\text{choose}}$ which provides him with a random tag that just entered the supply chain at step v_0. $\mathcal{A}$ then iterates the supply chain for a maximum of ρ times. At each iteration i, $\mathcal{A}$ reads out T's state and writes into T. Also, $\mathcal{O}_{\text{step}}$ provides $\mathcal{A}$ with the step $v_{T,(i+1)}$ that T will go through in the next supply chain iteration. $\mathcal{A}$ then queries the oracle $\mathcal{O}_{\text{draw}}$ with step $v_{T,(i+1)}$. $\mathcal{O}_{\text{draw}}$ provides $\mathcal{A}$ with r tags $T_{i,j}$ that will go through $v_{T,(i+1)}$ in the next supply chain iteration that he can read and write into. Also, $\mathcal{O}_{\text{select}}$ provides $\mathcal{A}$ with s tags $T'_{i,j}$ from $\mathcal{T}$ along with the steps they went through so far. $\mathcal{A}$ is also provided with the next step of tags $T'_{i,j}$ by calling the oracle $\mathcal{O}_{\text{step}}$. $\mathcal{A}$ then iterates the supply chain and reads the updated states of the r tags provided by $\mathcal{O}_{\text{draw}}$ and the s tags provided by $\mathcal{O}_{\text{select}}$.

As in the tag unlinkability game, this step unlinkability game reflects the capabilities of an active adversary who, besides eavesdropping on tags' communication, can as well follow tags and tamper with their internal states along different steps of the supply chain.

```
$T \leftarrow \mathcal{O}_{\text{choose}}$;
for $i := 0$ to $\rho - 1$ do
    $T \rightarrow \mathcal{O}_{\text{step}}$;
    $v_{T,(i+1)} \leftarrow \mathcal{O}_{\text{step}}$;
    $s_T^i$ :=READ($T$);
    WRITE($T, s_T'^i$);
    for $j := 1$ to $r$ do
        $v_{T,(i+1)} \rightarrow \mathcal{O}_{\text{draw}}$;
        $(T_{i,j}, S_{i,j}) \leftarrow \mathcal{O}_{\text{draw}}$;
        $s_{T_{i,j}}$ :=READ($T_{i,j}$);
        WRITE($T_{i,j}, s'_{T_{i,j}}$);
    end
    for $j := 1$ to $s$ do
        $(T'_{i,j}, S'_{i,j}) \leftarrow \mathcal{O}_{\text{select}}$;
        $T'_{i,j} \rightarrow \mathcal{O}_{\text{step}}$;
        $v_{T'_{i,j}} \leftarrow \mathcal{O}_{\text{step}}$;
        $s_{T'_{i,j}}$ :=READ($T'_{i,j}$);
        WRITE($T'_{i,j}, s'_{T'_{i,j}}$);
    end
    ITERATESUPPLYCHAIN;
    for $j := 1$ to $r$ do
        READ($T_{i,j}$);
    end
    for $j := 1$ to $s$ do
        READ($T'_{i,j}$);
    end
end
```

Algorithm 5:
$\mathcal{A}$'s step unlinkability learning phase

```
$T_c \leftarrow \mathcal{O}_{\text{choose}}$;
for $i := 0$ to $\rho - 1$ do
    $s_{T_c}^i$ :=READ($T_c$);
    WRITE($T_c, s_{T_c}'^i$);
    for $j := 1$ to $s$ do
        $(T_{i,j}, S_{i,j}) \leftarrow \mathcal{O}_{\text{select}}$;
        $s_{T_{i,j}}$ :=READ($T_{i,j}$);
        WRITE($T_{i,j}, s'_{T_{i,j}}$);
        $T_{i,j} \rightarrow \mathcal{O}_{\text{step}}$;
        $v_{T_{i,j}} \leftarrow \mathcal{O}_{\text{step}}$;
    end
    ITERATESUPPLYCHAIN;
    for $j := 1$ to $s$ do
        READ($T_{i,j}$);
    end
end
READ($T_c$);
OUTPUT $b$;
```

Algorithm 6:
$\mathcal{A}$'s step unlinkability challenge phase

In the challenge phase, cf., Algorithm 6, $\mathcal{A}$ is provided with a challenge tag T_c which just entered the supply chain. $\mathcal{A}$'s goal is to tell whether the paths that tag T and tag T_c took have a step in common – trivially, besides the initial step v_0. $\mathcal{A}$ iterates the supply chain for a maximum of ρ times. At each iteration i, $\mathcal{A}$ reads out and writes into T_c. $\mathcal{A}$ calls as well the oracle $\mathcal{O}_{\text{select}}$ that provides him with s tags $T_{i,j}$ which he can read and write into. He is also provided with the step $v_{T_{i,j}}$ that $T_{i,j}$ will go through in the next iteration. $\mathcal{A}$ then iterates the supply chain and reads the updated state of the s tags. At the end of the challenge phase, $\mathcal{A}$ reads the current state of tag T_c and outputs $b = 1$, if T_c and T have a step in common (besides v_0), and $b = 2$ if they do not have a step in common (besides v_0). The adversary is successful, if his guess is correct.

Definition 4 (Step Unlinkability). TRACKER *provides step unlinkability*$\Leftrightarrow$
For adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ outputs a correct guess$) \leq \frac{1}{2} + \epsilon$ holds, where ϵ is negligible.

The above definition covers unlinkability of individual steps in the supply chain. **Note** that "step unlinkability" is stronger than "unlinkability of paths" that prevents an adversary $\mathcal{A}$ from telling whether two tags went through the same path or not. If $\mathcal{A}$ is able to tell whether two tags went through the same path then he automatically knows that the paths of these two tags have steps in common. So, if TRACKER provides step unlinkability, it will as well provide path unlinkability. Step *unlinkability* also implies step and path *privacy*.

4 Tracker Protocol

Protocol overview: In TRACKER, a tag T's state s_T^l represents the sequence of steps in the supply chain that T went through. The main concept is to represent different paths in the supply chain using different *polynomials*. More precisely, at the end of a supply chain's valid path $\mathcal{P}_{\text{valid}}$, a tag's state s_T^l will match the evaluation of a unique polynomial $Q_{\mathcal{P}_{\text{valid}}}(x)$ in a fixed value x_0. Therefore, a path in the supply chain is represented by $Q_{\mathcal{P}_{\text{valid}}}(x_0) \in \mathbb{F}_q$ providing a compact and efficient representation of paths.

Now, TRACKER relies on the property that for any two *different* paths $\mathcal{P} \neq \mathcal{P}'$, valid or not, the equation $Q_{\mathcal{P}}(x_0) = Q_{\mathcal{P}'}(x_0)$ holds only with negligible probability. Two different paths will result in two different polynomial evaluations. As a result, the state of a tag T at the end of the supply chain can be uniquely related to one single (valid) path.

However, the path representation as presented above does not suffice to prevent path cloning, i.e., copying the path of a valid tag into a fake tag and then injecting the fake tag in the supply chain. To tackle this problem, tags will store $Q_{\mathcal{P}_{\text{valid}}}(x_0)$ multiplied by a keyed HMAC of their unique IDs. HMAC serves two purposes: first, it proves that tags are issued by a legitimate authority and prevents an adversary from injecting its own tags. Second, it allows to map the tag's ID to a random number that cannot be predicted by the adversary. A tag's state therefore consists of three elements that are: a unique ID, $\text{HMAC}_k(\text{ID})$ and $\text{HMAC}_k(\text{ID}) \cdot Q_{\mathcal{P}_{\text{valid}}}(x_0)$.

TRACKER can be structured into three parts: **1.)** Issuer I writes an initial state s_T^0 into a new tag T. **2.)** Readers successively compute the evaluation of a polynomial: to achieve the evaluation of the "entire" polynomial $Q_{\mathcal{P}_{\text{valid}}}(x_0)$ at the end of a valid path, each reader visited by tag T computes T's new state s_T^i by applying simple arithmetic operations represented by the function f_i on the T's current state s_T^{i-1}. Eventually, this results in the evaluation of the entire polynomial $Q_{\mathcal{P}_{\text{valid}}}(x_0)$. **3.)** Finally, manager M checks a tag's state s_T^l. M knows a set of ν evaluations of valid polynomials $Q_{\mathcal{P}_{\text{valid}_i}}(x_0)$. M checks whether one of these polynomials corresponds to s_T^l, and if so, M knows the path the tag has taken.

Privacy and security overview: On the one hand, to protect privacy (more precisely "unlinkability") in TRACKER, tags store only probabilistic elliptic curve Elgamal encryptions of their states, and readers use homomorphic (re-)encryption techniques for the arithmetic operations on encrypted path encodings. At the end of the supply chain, the manager can then decrypt and identify the path.

On the other hand, security of TRACKER relies on both the security of Elgamal and the security of HMAC. A tag stores an *encrypted* state of the three elements: ID, $\text{HMAC}_k(\text{ID})$ and $\text{HMAC}_k(\text{ID}) \cdot Q_{\mathcal{P}_{\text{valid}}}(x_0)$. Although an adversary can always have access to encryptions of $\text{HMAC}_k(\text{ID})$ and encryptions of $Q_{\mathcal{P}_{\text{valid}}}(x_0)$, he cannot come up with an encryption that corresponds to the product of the underlying plaintexts, that is, $\text{HMAC}_k(\text{ID}) \cdot Q_{\mathcal{P}_{\text{valid}}}(x_0)$. We show in Section 5.2 that if an adversary $\mathcal{A}$ is able to come up with an encryption of $\text{HMAC}_k(\text{ID}) \cdot Q_{\mathcal{P}_{\text{valid}}}(x_0)$, he will be able to break either computational Diffie-Hellman (CDH) or HMAC security. Before the detailed protocol description in Section 4.3, we will first provide an overview about TRACKER's polynomial path encoding and elliptic curve encryption used in this paper.

4.1 Path Encoding in Tracker

TRACKER's polynomial path encoding is based on techniques for software fault detection. Noubir et al. [19] propose to encode a software's state machine using polynomials such that the exact sequence of states visited during run-time generates a unique "mark". Therewith, run-time faults can be detected. TRACKER's path encoding is based on the one by Noubir et al. [19] and will be described in the following.

For each step $v_i, 1 \leq i \leq \eta$, in the supply chain, v_i is associated with a unique random number $a_i \in \mathbb{F}_q$, where q is a large prime. Accordingly, the issuer's step v_0 is associated with a random number $a_0 \in \mathbb{F}_q$.

As mentioned above, a path in the supply chain is represented as a polynomial in $\mathbb{F}_q$. The polynomial corresponding to a path $\mathcal{P} = \overrightarrow{v_0 v_1 \ldots v_l}$ is defined in Equation (1). All operations are in $\mathbb{F}_q$.

$$Q_{\mathcal{P}}(x) := a_0 x^l + \sum_{i=1}^{l} a_i x^{l-i}. \tag{1}$$

To have a more compact representation of paths, a path $\mathcal{P}$ is represented as the evaluation of $Q_{\mathcal{P}}(x)$ in x_0, where x_0 is a generator of $\mathbb{F}_q^*$. We denote $\phi(\mathcal{P}) = Q_{\mathcal{P}}(x_0)$. The above path encoding using polynomials with random coefficients $a_i \in \mathbb{F}_q$ has the desired property that any two different paths result in distinct polynomial evaluations with high probability. That is, $\forall \mathcal{P}, \mathcal{P}'$ with $\mathcal{P} \neq \mathcal{P}'$, equation $\phi(\mathcal{P}) = \phi(\mathcal{P}')$ holds with probability $\frac{1}{q}$, cf., Noubir et al. [19].

Let T be a tag with a unique ID that took path $\mathcal{P}$. We define T's *path mark* as: $\phi_{\mathrm{ID}}(\mathcal{P}) := \mathrm{HMAC}_k(\mathrm{ID}) \cdot \phi(\mathcal{P})$. As defined above, the path mark depends on tags' ID to prevent an adversary from copying the path mark of a tag into another one. Although the path mark depends on ID, knowledge of $\phi_{\mathrm{ID}}(\mathcal{P})$ and $\mathrm{HMAC}_k(\mathrm{ID})$ allows M to always derive $\phi(\mathcal{P})$ and identify $\mathcal{P}$.

Readers: The path mark $\phi_{\mathrm{ID}}(\mathcal{P})$ is stored on the tag. A reader that is visited by a tag T reads T's current path mark, updates it, and writes the updated path mark back into T. To eventually achieve the evaluation of path mark $\phi_{\mathrm{ID}}(\mathcal{P})$ of path $\mathcal{P} = \overrightarrow{v_0 v_1 \ldots v_{i-1} v_i v_{i+1} \ldots v_l}$, the per reader effort is quite low. Assume that T arrives at reader R_i, i.e., step v_i in the supply chain. So far, T went through (sub-)path $\mathcal{P}_{i-1} = \overrightarrow{v_0 v_1 \ldots v_{i-1}}$, and stores ID, $\mathrm{HMAC}_k(\mathrm{ID})$, and path mark $\phi_{\mathrm{ID}}(\mathcal{P}_{i-1})$.

To get $\phi_{\mathrm{ID}}(\mathcal{P}_i)$, reader R_i simply computes its state transition function f_{R_i} defined as

$$f_{R_i}(x) := x_0 x + \mathrm{HMAC}_k(\mathrm{ID}) \cdot a_i.$$

So, $\phi_{\mathrm{ID}}(\mathcal{P}_i) := f_{R_i}(\phi_{\mathrm{ID}}(\mathcal{P}_{i-1})) = x_0 \phi_{\mathrm{ID}}(\mathcal{P}_{i-1}) + \mathrm{HMAC}_k(\mathrm{ID}) \cdot a_i$. R_i writes $\phi_{\mathrm{ID}}(\mathcal{P}_i)$ in T. By construction, this will eventually result in $\phi_{\mathrm{ID}}(\mathcal{P}_i) = \mathrm{HMAC}_k(\mathrm{ID}) \cdot (a_0 x_0^l + \sum_{j=1}^{i} a_j x_0^{i-j}) = \mathrm{HMAC}_k(\mathrm{ID}) \cdot \phi(\mathcal{P}_i)$.
Tag state decoding: This operation corresponds to the CHECK function of the TRACKER protocol.

The state s_T^l of a valid tag T in the supply chain that went through a valid path $\mathcal{P}_{\mathrm{valid}}$ consists of a tuple of three elements $s_T^l := (\mathrm{ID}, \mathrm{HMAC}_k(\mathrm{ID}), \phi_{\mathrm{ID}}(\mathcal{P}_{\mathrm{valid}}))$.

Before decoding $\phi_{\mathrm{ID}}(\mathcal{P}_{\mathrm{valid}})$, M provided with the secret key k and ID, computes $\mathrm{HMAC}_k(\mathrm{ID})$ and verifies the second element of T's state. If T passes the verification, M multiplies $\phi_{\mathrm{ID}}(\mathcal{P}_{\mathrm{valid}})$ by $\mathrm{HMAC}_k(\mathrm{ID})^{-1}$ to get $\phi(\mathcal{P}_{\mathrm{valid}})$. M stores a list of all possible $\phi(\mathcal{P}_{\mathrm{valid}_i})$ along with their corresponding valid paths. Given $\phi(\mathcal{P}_{\mathrm{valid}})$, M will be able to check and identify the path $\mathcal{P}_{\mathrm{valid}}$.

As we will now see in the following paragraphs, tags in TRACKER store *encrypted* versions of ID, $\mathrm{HMAC}_k(\mathrm{ID})$ and $\phi_{\mathrm{ID}}(\mathcal{P}_{\mathrm{valid}})$. So in conclusion, a tag stores the tuple: $s_T^l = (E(\mathrm{ID}), E(\mathrm{HMAC}_k(\mathrm{ID})), E(\phi_{\mathrm{ID}}(\mathcal{P}_{\mathrm{valid}}))$.

4.2 Elliptic Curve Elgamal Cryptosystem

An elliptic curve Elgamal cryptosystem provides the following usual set of operations:

Setup: The system outputs an elliptic curve $\mathcal{E}$ over a finite field $\mathbb{F}_p$. Let P be a point on $\mathcal{E}(\mathbb{F}_p)$ of a large prime order q such that the discrete logarithm problem is intractable for $\mathcal{G} =< P >$. Here, p and q are TRACKER security parameters, e.g., $|p| = |q| = 160$ bit.

Key generation: The secret key is $sk \in \mathbb{F}_q$. The corresponding public key pk is the pair of points $(P, Y = sk \cdot P)$.

Encryption: To encrypt a point $M \in \mathcal{E}$, one randomly selects $r \in \mathbb{F}_q$ and computes $E(M) := (U, V) = (r \cdot P, M + r \cdot Y)$. The ciphertext is $c = (U, V)$.

Decryption: To decrypt a ciphertext $c = (U, V)$, one computes $D(c) := U - sk \cdot V = M$. In TRACKER, a tag in the supply chain stores the elliptic curve Elgamal encryption of its unique ID, $\mathrm{HMAC}_k(\mathrm{ID})$, and a path mark $\phi_{\mathrm{ID}}(\mathcal{P})$. Without loss of generality, we assume that ID of a tag is a random point in the elliptic curve $\mathcal{E}$ and that $\mathrm{HMAC}_k(\mathrm{ID})$ is an element of $\mathbb{F}_q$ such that $|q| = 160$ bits.

To encrypt $\mathrm{HMAC}_k(\mathrm{ID})$ and $\phi_{\mathrm{ID}}(\mathcal{P})$ in $\mathbb{F}_q$ using Elgamal over elliptic curves, we need a point mapping which transforms a message $m \in \mathbb{F}_q$ to a point in the elliptic curve $\mathcal{E}$.

Point mapping: We use a simple additively homomorphic mapping $\mathcal{M} : \mathbb{F}_q \rightarrow \mathcal{E}$ that preserves the properties of our polynomial path encoding with respect to the probability of path "collisions". Message $m \in \mathbb{F}_q$ is mapped to a point in $\mathcal{E}$ by $\mathcal{M}(m) = m \cdot P$, where P is a point in $\mathcal{E}$ of large prime order q.

This mapping is a one-to-one mapping from $\mathbb{F}_q$ to $\mathcal{G} =< P >$: if $\exists m_1, m_2 \in \mathbb{F}_q$ such that $\mathcal{M}(m_1) = \mathcal{M}(m_2)$, then $m_1 = m_2 \bmod q$. Therefore, the probability that the mappings of two path marks collide in $\mathcal{E}$, i.e., $\mathcal{M}(\phi_{\text{ID}}(\mathcal{P}_1)) = \mathcal{M}(\phi_{\text{ID}}(\mathcal{P}_2))$, is the same as the probability that two path marks collide in $\mathbb{F}_q$. This mapping is not reversible which means that we cannot deduce $\phi_{\text{ID}}(\mathcal{P})$ from $\mathcal{M}(\phi_{\text{ID}}(\mathcal{P}))$. However, this is not an issue in TRACKER: as mentioned above, the manager knows the valid paths in advance. So he computes and stores the *mappings* $\mathcal{M}(\phi(\mathcal{P}_{\text{valid}_i})) \in \mathcal{E}$, instead of computing and storing $\phi(\mathcal{P}_{\text{valid}_i}) \in \mathbb{F}_q$. Given ID, the manager computes $\text{HMAC}_k(\text{ID})$, derives the mapping $\mathcal{M}(\phi(\mathcal{P})) \in \mathcal{E}$ from $\mathcal{M}(\phi_{\text{ID}}(\mathcal{P}))$, and then checks if $\mathcal{P}$ is a valid path by comparing $\mathcal{M}(\phi(\mathcal{P}))$ with the list of valid mappings..

4.3 Detailed Protocol Description

TRACKER consists of an initial setup phase, the preparation of new tags entering the supply chain, reader and tag interaction as part of the supply chain, and finally a path verification conducted by manager M.

Tracker initialization: Issuer I sets up an elliptic curve Elgamal cryptosystem and generates the secret key sk and the public key $pk = (P, Y = sk \cdot P)$ such that the order of P is a large prime q, $|q| = 160$ bit.

Then, I selects x_0 a generator of the finite field $\mathbb{F}_q$, and selects randomly a value $a_0 \in \mathbb{F}_q$. I generates a random bit string k, $|k| = 160$ bit. The initial step v_0, representing the issuer in the supply chain, is associated with (a_0, k).

Similarly, I generates η random numbers $a_i \in \mathbb{F}_q, 1 \leq \eta$. I sends to each reader R_i, representing step v_i, the tuple (x_0, a_i) using a secure channel.

Also using a secure channel, I provides manager M with secret key sk, generator x_0, key k and tuples (i, a_i). Therewith, M is informed which reader R_i at step v_i knows which a_i. As M knows which paths in the supply chain will be valid, he now computes all the ν valid $\phi(\mathcal{P}_{\text{valid}})$ using Equation (1). Finally, M computes and stores pairs $(\mathcal{M}(\phi(\mathcal{P}_{\text{valid}_i})), \text{steps})$, where steps is the sequence of steps $\overrightarrow{v_0 v_{\mathcal{P}_{\text{valid}},1} v_{\mathcal{P}_{\text{valid}},2} \cdots v_{\mathcal{P}_{\text{valid}},l}}$ of $\mathcal{P}_{\text{valid}_i}$. That is, M knows for each mapping the sequence of steps. Therefore, the manager verifies the validity of the path and if the path is valid he can identify it.

In conclusion, x_0 is public, the a_i are secret and only known by reader R_i and M. Also, only M and I know sk and k.

Tag preparation: For each new tag T entering the supply chain, I draws a random point $\text{ID} \in \mathcal{E}$ which is T's unique identifier. Now, let HMAC_k be a (secure) HMAC algorithm [5], $\text{HMAC}_k(m) : \mathbb{F}_q \times \mathcal{E} \rightarrow \mathbb{F}_q$. Provided with key k, Issuer computes $\text{HMAC}_k(\text{ID})$.

I then selects three random numbers $r^0_{\text{ID}}, r^0_\sigma, r^0_\phi \in \mathbb{F}_q$ to compute the following ciphertexts:

$$\begin{aligned} c^0_{\text{ID}} &= E(\text{ID}) = (U^0_{\text{ID}}, V^0_{\text{ID}}) = (r^0_{\text{ID}} \cdot P, \text{ID} + r^0_{\text{ID}} \cdot Y) \\ c^0_\sigma &= E(\text{HMAC}_k(\text{ID})) = (U^0_\sigma, V^0_\sigma) = (r^0_\sigma \cdot P, \text{HMAC}_k(\text{ID}) \cdot P + r^0_\sigma \cdot Y) \\ c^0_\phi &= E(\phi_{\text{ID}}(v_0)) = (U^0_\phi, V^0_\phi) = (r^0_\phi \cdot P, \text{HMAC}_k(\text{ID}) \cdot a_0 \cdot P + r^0_\phi \cdot Y) \end{aligned}$$

Finally, I writes state $s^0_T = (c^0_{\text{ID}}, c^0_\sigma, c^0_\phi)$ into T that can enter the supply chain.

Tag and reader interaction in the supply chain: Assume a tag T arrives at step v_i and reader R_i in the supply chain. Without loss of generality, assume that the path that tag T took so far is $\mathcal{P}_{i-1} = \overrightarrow{v_0 v_1 \ldots v_{i-1}}$ and let $\mathcal{P}_i = \overrightarrow{v_0 v_1 \ldots v_i}$. R_i reads out T's current state $s^{i-1}_T = (c^{i-1}_{\text{ID}}, c^{i-1}_\sigma, c^{i-1}_\phi)$.

Given the ciphertexts $c^{i-1}_\phi = (U^{i-1}_\phi, V^{i-1}_\phi)$, $c^{i-1}_\sigma = (U^{i-1}_\sigma, V^{i-1}_\sigma)$, generator x_0, and a_i, R_i computes $c^i_\phi = (U^i_\phi, V^i_\phi)$:

$$
\begin{aligned}
U_\phi^i &= x_0 \cdot U_\phi^{i-1} + a_i \cdot U_\sigma^{i-1} = (x_0 r_\phi^{i-1} + a_i r_\sigma^{i-1}) \cdot P \\
V_\phi^i &= x_0 \cdot V_\phi^{i-1} + a_i \cdot V_\sigma^{i-1} \\
&= x_0 (\sum_{j=0}^{i-1} \mathrm{HMAC}_k(\mathrm{ID}) \cdot a_j x_0^{i-1-j} \cdot P + r_\phi^{i-1} \cdot Y) + a_i (\mathrm{HMAC}_k(\mathrm{ID}) \cdot P + r_\sigma^{i-1} \cdot Y) \\
&= \sum_{j=0}^{i-1} \mathrm{HMAC}_k(\mathrm{ID}) \cdot a_j x_0^{i-1-j} \cdot P + \mathrm{HMAC}_k(\mathrm{ID}) \cdot a_i \cdot P + x_0 r_\phi^{i-1} \cdot Y + a_i r_\sigma^{i-1} \cdot Y \\
&= \mathrm{HMAC}_k(\mathrm{ID}) \cdot \sum_{j=0}^{i} a_j x_0^{i-j} \cdot P + (x_0 r_\phi^{i-1} + a_i r_\sigma^{i-1}) \cdot Y \\
&= \mathcal{M}(\mathrm{HMAC}_k(\mathrm{ID}) \cdot \phi(\mathcal{P}_i)) + (x_0 r_\phi^{i-1} + a_i r_\sigma^{i-1}) \cdot Y = \mathcal{M}(\phi_{\mathrm{ID}}(\mathcal{P}_i)) + (x_0 r_\phi^{i-1} + a_i r_\sigma^{i-1}) \cdot Y
\end{aligned}
$$

In conclusion, the above is the homomorphic encryption variant of the reader computation of Section 4.1.

To get c_{ID}^i and c_σ^i, reader R_i re-encrypts c_{ID}^{i-1} and c_σ^{i-1}, respectively: it picks randomly two numbers r'_{ID} and $r'_\sigma \in \mathbb{F}_q$ and outputs two new ciphertexts $c_{\mathrm{ID}}^i = (U_{\mathrm{ID}}^i, V_{\mathrm{ID}}^i) = (r'_{\mathrm{ID}} \cdot P + U_{\mathrm{ID}}^{i-1}, r'_{\mathrm{ID}} \cdot Y + V_{\mathrm{ID}}^i)$ and $c_\sigma^i = (U_\sigma^i, V_\sigma^i) = (r'_\sigma \cdot P + U_\sigma^{i-1}, r'_\sigma \cdot Y + V_\sigma^i)$.

The reader also re-encrypts c_ϕ^i. It picks randomly $r'_\phi \in \mathbb{F}_q$ and outputs: $c_\phi'^i = (U_\phi'^i, V_\phi'^i) = (r'_\phi \cdot P + U_\phi^i, r'_\phi \cdot Y + V_\phi^i)$. Finally, R_i writes the new state $s_T^i = (c_{\mathrm{ID}}^i, c_\sigma^i, c_\phi'^i)$ into T.

Path verification by M: This operation corresponds to TRACKER's realization of the CHECK function. Upon reading a tag's state $s_T^l = (c_{\mathrm{ID}}^l, c_\sigma^l, c_\phi^l)$, M decrypts c_{ID}^l and gets $\mathrm{ID} \in \mathcal{E}$. M checks then for cloning by looking up ID in M's database $\mathrm{DB}_{\mathrm{clone}}$. If $\mathrm{ID} \in \mathrm{DB}_{\mathrm{clone}}$, then M outputs $\emptyset$ and rejects T.

Otherwise, M decrypts c_σ^l to get a point $Q \in \mathcal{E}$. M computes $\mathrm{HMAC}_k(\mathrm{ID})$ and $\mathcal{M}(\mathrm{HMAC}_k(\mathrm{ID}))$, and verifies whether the equation $Q = \mathcal{M}(\mathrm{HMAC}_k(\mathrm{ID}))$ holds. If it does not, M outputs $\emptyset$ and rejects T. If $Q = \mathcal{M}(\mathrm{HMAC}_k(\mathrm{ID}))$, M decrypts c_ϕ^l which results in a point Q'. Given $\mathrm{HMAC}_k(\mathrm{ID})$, M computes the inverse of $\mathrm{HMAC}_k(\mathrm{ID}) \in \mathbb{F}_q$, and then computes $\pi = \mathrm{HMAC}_k(\mathrm{ID})^{-1} \cdot Q' = \mathcal{M}(\phi(\mathcal{P}))$.

M checks, whether π is in his list of valid mappings $\mathcal{M}(\phi(\mathcal{P}_{\mathrm{valid}_i}))$. If there is no match, M outputs $\emptyset$ and rejects the tag. Otherwise, manager M outputs $\mathcal{P}_{\mathrm{valid}}$ and adds ID to $\mathrm{DB}_{\mathrm{clone}}$.

5 Security and Privacy Analysis

Before giving the security and the privacy analysis, we introduce the security properties of HMAC.

5.1 HMAC Security

An HMAC with key k, a message m, and a cryptographic hash function h is defined as $\mathrm{HMAC}_k(m) := h(k \oplus \mathrm{opad} || h(k \oplus \mathrm{ipad} || m))$, where $||$ is concatenation. For more details about opad and ipad see Krawczyk et al. [16].

If the output of h and the secret key k are indistinguishable from random data for an adversary, then HMAC_k holds the following two properties [4, 5]:

1.) Resistance to existential forgery: Let $\mathcal{O}_{\mathrm{HMAC}_k}^{\mathrm{forge}}$ be an HMAC oracle that, when provided with a message m, returns $\mathrm{HMAC}_k(m)$. An adversary $\mathcal{A}$ can choose N messages $m_1, \ldots, m_N$, and provide them to the oracle $\mathcal{O}_{\mathrm{HMAC}_k}^{\mathrm{forge}}$ to get the corresponding $\mathrm{HMAC}_k(m_i)$. Still, the advantage ϵ of $\mathcal{A}$ to come up with a new pair $(m, \mathrm{HMAC}_k(m))$, where $m \neq m_i, 1 \leq i \leq N$, is negligible.

2.) Indistinguishability: Let $\mathcal{O}_{\mathrm{HMAC}_k}^{\mathrm{distinguish}}$ be an HMAC oracle, when provided with a message m, it flips a coin $b \in \{0, 1\}$ and returns a message m' such that: if b = 0, it returns a random number. If b = 1, it returns

$\mathrm{HMAC}_k(m)$. Even knowing m, $\mathcal{A}$ cannot tell if m' is a random number or $m' = \mathrm{HMAC}_k(m)$. That is, HMAC_k is a pseudo-random function.

5.2 Security

First, an adversary $\mathcal{A}$ winning the security game of Algorithm 2 implies that $\mathcal{A}$ writes into a tag T a valid state $s_T = (c'_{\mathrm{ID}}, c'_\sigma, c'_\phi)$. This implies that the pair $(c'_{\mathrm{ID}}, c'_\sigma)$ is a valid pair, i.e., $c'_{\mathrm{ID}} = E(\mathrm{ID})$ and $c'_\sigma = E(\mathrm{HMAC}_k(\mathrm{ID}) \cdot P)$. Producing a *new* valid pair $(c'_{\mathrm{ID}}, c'_\sigma)$ entails that $\mathcal{A}$ is breaking the security of HMAC as sketched in Lemma 1.

Note: We say that $\mathcal{A}$ produces a new valid pair $(c'_{\mathrm{ID}} = E(\mathrm{ID}), c'_\sigma = E(\mathrm{HMAC}_k(\mathrm{ID}) \cdot P))$, if $(c'_{\mathrm{ID}}, c'_\sigma)$ is not (a re-encryption of) a pair $(c_{\mathrm{ID}}, c_\sigma)$ that $\mathcal{A}$ read during the learning phase.

Lemma 1. *Producing a new valid pair $(c'_{\mathrm{ID}}, c'_\sigma)$ contradicts the indistinguishability property of HMAC.*

Proof (Sketch). More precisely, we can build an adversary $\mathcal{A}'$ that uses $\mathcal{A}$ to break the indistinguishability property of HMAC_k. When $\mathcal{A}$ provides $\mathcal{A}'$ with a new pair $(c'_{\mathrm{ID}}, c'_\sigma)$, $\mathcal{A}'$ decrypts c'_{ID} and c'_σ and gets ID and a point Q respectively. $\mathcal{A}'$ gives ID to $\mathcal{O}^{\mathrm{distinguish}}_{\mathrm{HMAC}_k}$. $\mathcal{O}^{\mathrm{distinguish}}_{\mathrm{HMAC}_k}$ returns a message m'. Finally, to break the indistinguishability of HMAC_k, $\mathcal{A}'$ checks whether $Q = m' \cdot P$. If so, $\mathcal{A}'$ outputs 1, meaning that $m' = \mathrm{HMAC}_k(\mathrm{ID})$. Otherwise, $\mathcal{A}'$ outputs 0 implying that m' is a random number. □

Theorem 1. TRACKER *is secure under the security of HMAC and the computational Diffie-Hellman (CDH) assumption.*

Proof. As of Lemma 1, $\mathcal{A}$ cannot compute a new valid pair $(c'_{\mathrm{ID}}, c'_\sigma)$. If $\mathcal{A}$ re-uses a valid pair $(c'_{\mathrm{ID}}, c'_\sigma)$ read in the learning phase, then providing a valid tuple $(c'_{\mathrm{ID}}, c'_\sigma, c'_\phi)$ implies that $\mathcal{A}$ is able to solve an instance of the computational Diffie-Hellman problem as shown below.

Assume there would be an adversary $\mathcal{A}(\rho, r, \epsilon)$ that breaks the security of TRACKER by choosing arbitrarily a tag $T \in \mathcal{T}$, then re-writing it with a valid state $(c'_{\mathrm{ID}}, c'_\sigma, c'_\phi)$. If this is the case, and if the output of HMAC is indistinguishable from a random number, we show that there is an adversary $\mathcal{A}'$ that breaks the CDH assumption with non-negligible advantage ϵ'.

Note that we do not cover simple cloning here, as an adversary can always succeed in copying the state of a tag that went through a valid path. As discussed before, anti-cloning protection is provided by $\mathrm{DB}_{\mathrm{clone}}$.

Let $\mathcal{O}_{\mathrm{CDH}}$ be an oracle that, when it is queried, selects randomly two elements a and b in $\mathbb{F}_q$ and returns the tuple $(P, a \cdot P, b \cdot P)$. An adversary $\mathcal{A}'$ breaks CDH, if given $(P, a \cdot P, b \cdot P)$, he outputs $ab \cdot P$.

Overview: In a nutshell, an adversary $\mathcal{A}$ is able to break TRACKER, if he outputs an encryption of $\mathrm{ID}, \mathrm{HMAC}_k(\mathrm{ID}) \cdot P$, and $\mathrm{HMAC}_k(\mathrm{ID}) \cdot \phi(\mathcal{P}_{valid}) \cdot P$ from an encryption of $\mathrm{ID}, \mathrm{HMAC}_k(\mathrm{ID}) \cdot P$, and $\phi(\mathcal{P}_{valid}) \cdot P$. So to break CDH, $\mathcal{A}'$ uses $\mathcal{A}$ as a subroutine as follows: firstly, $\mathcal{A}'$ creates a TRACKER system with a valid path $\mathcal{P}_{\mathrm{valid}} = \overrightarrow{v_0 v_1 \ldots v_l}$. He randomly generates $(l-1)$ elements $a_i \in \mathbb{F}_q$ such that a_i corresponds to step v_i. The step v_l, however, will be associated with a point $R = b \cdot P - x_0 \sum_{i=0}^{l-1} a_i x_0^{l-1-i} \cdot P$. Therefore, $\mathcal{M}(\phi(\mathcal{P}_{\mathrm{valid}})) = b \cdot P$. Secondly, $\mathcal{A}'$ writes into a challenge tag T_n a state $s_{T_n} = (c_{\mathrm{ID}_n}, c_{\sigma_n}, c_{\phi_n})$ such that $c_{\sigma_n} = E(a \cdot P)$. If in the challenge phase of the security game $\mathcal{A}$ is able to write a valid state $(c'_{\mathrm{ID}_n}, c'_{\sigma_n}, c'_{\phi_n})$ into T_n which corresponds to the path $\mathcal{P}_{\mathrm{valid}}$, then $\mathcal{A}'$ will be able to break CDH by decrypting c'_{ϕ_n}. By construction, the path mark stored on T_n will correspond to $ab \cdot P$.

Details: For ease of understanding, we assume that the supply chain consists of only one valid path $\mathcal{P}_{\mathrm{valid}} = \overrightarrow{v_0 v_1 \ldots v_l}$ such that $\mathcal{M}(\phi(\mathcal{P}_{\mathrm{valid}})) = b \cdot P$.

- $\mathcal{A}'$ creates a TRACKER system with one valid path $\mathcal{P}_{\mathrm{valid}} = \overrightarrow{v_0 v_1 \ldots v_l}$: he generates randomly l elements $a_i, 0 \le i \le l-1$, such that a_i corresponds to v_i, the step v_l however is associated with a point $R = b \cdot P - x_0 \sum_{i=0}^{l-1} a_i x_0^{l-1-i} \cdot P$. Finally, $\mathcal{A}'$ generates a valid pair of keys (sk, pk) for Elgamal encryption and a key k for the HMAC.

– $\mathcal{A}'$ initializes $(n-1)$ tags T_i in TRACKER.
– To create the n^{th} tag T_n, $\mathcal{A}'$ picks randomly $\text{ID}_n \in \mathcal{E}$ and encrypts it into c_{ID_n}. Then, to compute c_{σ_n}, $\mathcal{A}'$ encrypts $a \cdot P$ instead of encrypting $\text{HMAC}_k(\text{ID}_n) \cdot P$. Given the indistinguishability property of HMAC, $\mathcal{A}$ cannot tell, whether $\mathcal{A}'$ computes the HMAC correctly or not. Finally, $\mathcal{A}'$ computes $c_{\phi_n} = E(aa_0 \cdot P)$.
– $\mathcal{A}'$ calls $\mathcal{A}(\rho, r, \epsilon)$ that enters the learning phase. $\mathcal{A}$ iterates the supply chain ρ times. At each iteration of the supply chain:
  1. $\mathcal{A}'$ updates the state of tags in the supply chain as follows: **first**, if a tag T_i is at step $v_i \neq v_l$, the tag is updated according to the TRACKER protocol. **Second**, if a tag T_i is at step v_l: $\mathcal{A}'$ decrypts the state $s_{T_i} = (c_{\text{ID}_i}, c_{\sigma_i}, c_{\phi_i})$ and gets three points $(\text{ID}'_i, Q_i, Q'_i)$. $\mathcal{A}$ checks whether $(\text{ID}'_i, Q_i, Q'_i)$ corresponds to a valid state of a tag going through the sub-path $\overrightarrow{v_0 v_1 \dots v_{l-1}}$, i.e., $Q_i = \text{HMAC}_k(\text{ID}'_i) \cdot P$ and $Q'_i = \text{HMAC}_k(\text{ID}'_i) \cdot \sum_{i=0}^{l-1} a_i x_0^{l-1-i} \cdot P$. If it is the case, $\mathcal{A}'$ writes into T_i a state $s'_{T_i} = (c'_{\text{ID}_i}, c'_{\sigma_i}, c'_{\phi_i})$ such that $c'_{\phi_i} = E(\text{HMAC}_k(\text{ID}'_i) \cdot b \cdot P)$. Otherwise, $\mathcal{A}'$ writes into T_i a state $s'_{T_i} = (c'_{\text{ID}_i}, c'_{\sigma_i}, c'_{\phi_i})$ such that c'_{ϕ_i} is an encryption of a random number.
     Note. Writing the encryption of a random number into an invalid tag T does not affect the output of the CHECK function. An invalid tag T either did not go through the valid sub-path $\overrightarrow{v_0 v_1 \dots v_{l-1}}$ or it stores an invalid HMAC. When $\mathcal{A}$ calls the CHECK function on T, CHECK will always output $\emptyset$. Moreover, a valid tag T that went through $\mathcal{P}_{\text{valid}}$ will always store a valid path mark corresponding to $\text{HMAC}_k(\text{ID}) \cdot b \cdot P$.
  2. Simulating $\mathcal{O}_{\text{pick}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with r tags that $\mathcal{A}$ is allowed to read from and write into.
  3. $\mathcal{A}$ gives back the r tags to $\mathcal{A}'$. $\mathcal{A}$ simulates $\mathcal{O}_{\text{M}}$ as follows:
     • Upon reading the state $s_{T'_i} = (c_{\text{ID}'_i}, c_{\sigma'_i}, c_{\phi'_i})$ of a tag T'_i, $\mathcal{A}'$ decrypts $c_{\text{ID}'_i}$ to get ID'_i. First, $\mathcal{A}'$ verifies whether $\text{ID}'_i = \text{ID}_n$. If it is the case, $\mathcal{A}'$ aborts and restarts the game. Otherwise, $\mathcal{A}$ decrypts $c_{\sigma'_i}$ and gets a point Q_i. Then, $\mathcal{A}'$ verifies whether the equation $Q_i = \text{HMAC}_k(\text{ID}'_i) \cdot P$ holds. If it does not hold, $\mathcal{A}'$ rejects the tag T'_i.
     • If $Q_i = \text{HMAC}_k(\text{ID}'_i) \cdot P$, $\mathcal{A}'$ decrypts $c_{\phi'_i}$ and gets a point Q'_i. $\mathcal{A}'$ then computes $\pi_i = \text{HMAC}_k(\text{ID}'_i)^{-1} \cdot Q'_i$. If $\pi_i = b \cdot P$, i.e., π_i is valid, $\mathcal{A}'$ outputs $\mathcal{P}_{\text{valid}}$. Otherwise, $\mathcal{A}'$ rejects the tag T'_i and outputs $\emptyset$.
– After the learning phase, $\mathcal{A}'$ puts $\mathcal{A}$ into the challenge phase. $\mathcal{A}$ then returns a tag $T \in \mathcal{T}$ which stores the state $(c'_{\text{ID}}, c'_{\sigma}, c'_{\phi})$ to $\mathcal{A}'$.

Once $\mathcal{A}'$ receives $(c'_{\text{ID}}, c'_{\sigma}, c'_{\phi})$, he decrypts c'_{ID} and gets ID using Elgamal secret key sk. He checks whether $\text{ID} = \text{ID}_n$, i.e., $T = T_n$. If it is not the case, $\mathcal{A}'$ restarts the game. Otherwise, $\mathcal{A}'$ decrypts c'_{ϕ}. Since $\mathcal{A}'$ computes the HMAC of ID_n as if it was a, the decryption of c'_{ϕ} results in a point $Q' = a \cdot \phi(\mathcal{P}_{\text{valid}}) \cdot P = ab \cdot P$. To solve the CDH problem $\mathcal{A}'$ outputs Q'.

$\mathcal{A}'$ succeeds in its attacks if: **1.)** the game does not abort: $\mathcal{A}$ is not provided with tag T_n in the learning phase. **2.)** In the challenge phase, $\mathcal{A}$ picks T_n.

In the learning phase, $\mathcal{A}$ is provided with $r \cdot \rho$ tags. Since tags are selected randomly among n tags, the probability that $\mathcal{A}$ is not provided with T_n in the learning phase is $(1 - \frac{1}{n})^{r \cdot \rho}$. Moreover, the probability that $\mathcal{A}$ picks T_n in the challenge phase is $\frac{1}{n}$. Therefore, if $\mathcal{A}(\rho, r, \epsilon)$ breaks TRACKER's security, then $\mathcal{A}'$ breaks CDH with advantage $\epsilon' = \frac{1}{n}(1 - \frac{1}{n})^{r \cdot \rho} \epsilon$. □

Above, we have shown that if there is an adversary $\mathcal{A}$ who breaks the security of TRACKER with one valid path, then there is an adversary who breaks CDH assumption. However, note that the security of TRACKER with one valid path can be extended to the security of TRACKER with multiple valid paths.

Lemma 2. *If there is an adversary $\mathcal{A}(\rho, r, \epsilon)$ who breaks* TRACKER*'s security with ν valid path, then there is an adversary $\mathcal{A}(\rho, r, \epsilon')$ who breaks* TRACKER*'s security with one valid path.*

Proof (sketch). In order to break TRACKER with one valid path $\mathcal{P}_{\text{valid}}$, $\mathcal{A}$ creates a supply chain of ν valid paths such that $\mathcal{P}_{\text{valid}}$ is one of the valid paths. Since $\mathcal{A}(\rho, r, \epsilon)$ breaks TRACKER with ν valid paths, he may output a tuple $(c'_{\text{ID}}, c'_{\phi}, \sigma'_i)$ that corresponds to the path $\mathcal{P}_{\text{valid}}$ with probability $\frac{1}{\nu}\epsilon$. Therefore, the advantage of $\mathcal{A}'$ is $\epsilon' = \frac{\epsilon}{\nu}$. □

In conclusion, if there is an adversary $\mathcal{A}(\rho, r, \epsilon)$ that breaks the security of TRACKER with ν valid paths, then there is an adversary $\mathcal{A}'$ who breaks CDH with advantage $\epsilon' = \frac{1}{\nu n}(1 - \frac{1}{n})^{r \cdot \rho}\epsilon$.

5.3 Privacy Analysis

For the privacy analysis, we use the semantic security property of Elgamal under re-encryption, cf., Golle et al. [10], to prove both tag unlinkability and step unlinkability.

Let $\mathcal{O}_{\text{re-encrypt}}$ be the oracle that, provided with two ciphertexts c_1, c_2, randomly chooses $b \in \{1, 2\}$, re-encrypts c_b using Elgamal and public key pk, and returns the resulting ciphertext c'_b.

As this re-encryption is based on Elgamal, the semantic security property of Elgamal encryption is extended to semantic security under re-encryption. Let $\mathcal{A}$ be an adversary that selects two ciphertexts c_1, c_2 and provides oracle $\mathcal{O}_{\text{re-encrypt}}$ with c_1 and c_2. $\mathcal{O}_{\text{re-encrypt}}$ randomly chooses b, re-encrypts c_b to c'_b, and returns c'_b to $\mathcal{A}$. The semantic security of Elgamal under re-encryption implies that guessing the value of b is as difficult for $\mathcal{A}$ as the decisional Diffie-Hellman (DDH) problem [10].

Theorem 2 (Tag Unlinkability). TRACKER *provides tag unlinkability under the DDH assumption.*

Proof. Assume there is an adversary $\mathcal{A}$ whose advantage ϵ to break the tag unlinkability experiment is non-negligible. We now construct a new adversary $\mathcal{A}'$ that executes $\mathcal{A}$ and breaks the semantic security of Elgamal under re-encryption ensured under the DDH assumption:

- $\mathcal{A}'$ creates a supply chain for the TRACKER protocol.
- $\mathcal{A}'$ calls the adversary $\mathcal{A}$. Simulating $\mathcal{O}_{\text{select}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with two pairs (T_1, S_1) and (T_2, S_2) such that T_1 and T_2 are selected randomly among the n tags in the supply chain, and S_1 (respectively S_2) is the set of steps that T_1 (respectively T_2) went through so far.
- $\mathcal{A}$ iterates the supply chain ρ times. At each iteration i of the supply chain:
  1. $\mathcal{A}$ reads and writes into T_1 and T_2.
  2. Simulating $\mathcal{O}_{\text{step}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with the next step that T_1 (respectively T_2) will go through in the next supply chain iteration.
  3. $\mathcal{A}'$ simulates $\mathcal{O}_{\text{select}}$ and provides $\mathcal{A}$ with r pairs $(T_{i,j}, S_{i,j}), 1 \leq j \leq r$, where $T_{i,j}$ is selected randomly, and $S_{i,j}$ is the set of steps that $T_{i,j}$ went through so far. $\mathcal{A}'$ is allowed to read from and write into these r tags.
- After the learning phase, $\mathcal{A}$ submits T_1 and T_2 to $\mathcal{A}'$ that simulates $\mathcal{O}_{\text{flip}}$. T_1 contains state $s_{T_1} = (c_{\text{ID}_1}, c_{\sigma 1}, c_{\phi_1})$, and T_2 contains state $s_{T_1} = (c_{\text{ID}_2}, c_{\sigma 2}, c_{\phi_2})$.
- $\mathcal{A}'$ transmits c_{ID_1} and c_{ID_2} to oracle $\mathcal{O}_{\text{re-encrypt}}$.
- $\mathcal{O}_{\text{re-encrypt}}$ randomly chooses b and returns the result c'_{ID_b} of re-encrypting one of the ciphertexts $c_{\text{ID}_1}, c_{\text{ID}_2}$ to $\mathcal{A}'$.
- $\mathcal{A}'$ prepares the challenge tag T_c:
  1. $\mathcal{A}'$ iterates the supply chain one more time.
  2. $\mathcal{A}'$ randomly selects $b' \in \{1, 2\}$ and stores the state $s_{T_c} = (c'_{\text{ID}_b}, c'_{\sigma_{b'}}, c'_{\phi_{b'}})$ in T_c.
- Simulating $\mathcal{O}_{\text{flip}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with the challenge tag T_c.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\text{select}}$ and provides $\mathcal{A}$ with s pairs $(T'_i, S'_i), 1 \leq i \leq s$, where T'_i is selected randomly, and S'_i is the set of steps that T'_i went through so far. $\mathcal{A}$ is allowed to read from and write into these s tags.

In general, given two events $\{E_1, E_2\}$, the probability that event E_1 occurs is always $Pr(E_1) = Pr(E_1|E_2) \cdot Pr(E_2) + Pr(E_1|\overline{E_2}) \cdot Pr(\overline{E_2})$. Now let E_1 be the event that $\mathcal{A}'$ can break the semantic security of Elgamal under re-encryption, and E_2 is the event that $b = b'$ holds.

If $b = b'$, the state $s_{T_c} = (c'_{\mathrm{ID}_b}, c'_{\sigma_{b'}}, c'_{\phi_{b'}})$ stored on T_c corresponds to a well formed tuple. Therefore, $\mathcal{A}$ outputs his guess for the tag corresponding to challenge tag T_c with non-negligible advantage ϵ. If $\mathcal{A}$ outputs T_1, this means that T_c stores a re-encryption of c_{ID_1}, and $\mathcal{A}'$ outputs 1. If $\mathcal{A}$ outputs T_2, this means that T_c stores a re-encryption of c_{ID_2}, and $\mathcal{A}'$ outputs 2.

If $b \neq b'$, the probability that $\mathcal{A}'$ breaks the semantic security of Elgamal under re-encryption is at worst a random guess, i.e., $\frac{1}{2}$.

Since b' is selected randomly, the probability that $b = b'$ holds is $\frac{1}{2}$. Therefore,

$$Pr(E_1) = Pr(E_1 \cap E_2) + Pr(E_1 \cap \overline{E_2}) = Pr(E_2) \cdot Pr(E_1|E_2) + Pr(\overline{E_2}) \cdot Pr(E_1|\overline{E_2})$$
$$= \frac{1}{2}Pr(E_1|E_2) + \frac{1}{2}Pr(E_1|\overline{E_2}) = \frac{1}{2}(\frac{1}{2} + \epsilon) + \frac{1}{2}Pr(E_1|\overline{E_2}) \geq \frac{1}{2}(\frac{1}{2} + \epsilon + \frac{1}{2}) = \frac{1}{2} + \frac{\epsilon}{2}$$

Consequently, the *advantage* of $\mathcal{A}'$ to break the semantic security of Elgamal under re-encryption is at least $\frac{\epsilon}{2}$. As a conclusion, if $\mathcal{A}$ has a non-negligible advantage ϵ to break TRACKER, $\mathcal{A}'$ as well will have a non-negligible advantage $\frac{\epsilon}{2}$ to break the semantic security of Elgamal under re-encryption.

Theorem 3 (Step Unlinkability). TRACKER *provides step unlinkability under the DDH assumption.*

Proof. Assume there is an adversary $\mathcal{A}$ whose advantage ϵ to break the step unlinkability experiment is non-negligible. We now construct a new adversary $\mathcal{A}'$ that executes $\mathcal{A}$ and breaks the semantic security of Elgamal under re-encryption:

- $\mathcal{A}'$ creates a supply chain for the TRACKER protocol with n tags, $\eta + 1$ steps, and ν valid paths.
- $\mathcal{A}'$ calls the adversary $\mathcal{A}$. Simulating $\mathcal{O}_{\text{choose}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with a tag T entering the supply chain.
- $\mathcal{A}$ iterates the supply chain ρ times. At each iteration i of the supply chain:
  1. $\mathcal{A}$ reads from and writes into T.
  2. Simulating $\mathcal{O}_{\text{step}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with the next step $v_{T,(i+1)}$ that T will go through in the next supply chain iteration.
  3. $\mathcal{A}'$ simulates $\mathcal{O}_{\text{draw}}$ and provides $\mathcal{A}$ with r pairs $(T_{i,j}, S_{i,j}), 1 \leq j \leq r$, where $T_{i,j}$ is a tag that will go through $v_{T,(i+1)}$ in the next iteration, and $S_{i,j}$ is the set of steps that $T_{i,j}$ went through so far. $\mathcal{A}$ is allowed to read from and write into these r tags.
  4. $\mathcal{A}'$ simulates $\mathcal{O}_{\text{select}}$ and provides $\mathcal{A}$ with s pairs $(T'_{i,j}, S'_{i,j}), 1 \leq j \leq s$, where $T'_{i,j}$ is a tag selected randomly, and $S'_{i,j}$ is the set of steps that $T'_{i,j}$ went through so far. $\mathcal{A}$ is allowed to read from and write into these s tags.
  5. $\mathcal{A}$ provides the oracle $\mathcal{O}_{\text{step}}$ with tags $T'_{i,j}$. $\mathcal{A}'$ simulates $\mathcal{O}_{\text{step}}$ and provides $\mathcal{A}$ with the next step of tags $T'_{i,j}$.
  6. When $\mathcal{A}$ iterates the supply chain, he will again receive the tags $T_{i,j}, T'_{i,j}$ which he can read from.

  Without loss of generality, we assume that T went through path $\mathcal{P} = \overrightarrow{v_0 v_1 \ldots v_\rho}$. Let $\mathcal{P}' = \overrightarrow{v_0 v'_1 \ldots v'_\rho}$ be a path such that $\mathcal{P}$ and $\mathcal{P}'$ have no step in common except for v_0.
- In the challenge phase, $\mathcal{A}'$ provides $\mathcal{A}$ with a challenge tag T_c that just entered the supply chain.
- $\mathcal{A}$ is allowed to iterate the supply chain ρ times.
- Before each iteration i:
  1. $\mathcal{A}$ can read from and write into T_c.
  2. $\mathcal{A}'$ simulates $\mathcal{O}_{\text{select}}$ and provides $\mathcal{A}$ with s pairs $(T'_{i,j}, S'_{i,j}), 1 \leq j \leq s$, where $T'_{i,j}$ is a tag selected randomly, and $S'_{i,j}$ is the set of steps that $T'_{i,j}$ went through so far. $\mathcal{A}'$ is allowed to read and write into these s tags.

3. $\mathcal{A}$ provides the oracle $\mathcal{O}_{\text{step}}$ with tags $T'_{i,j}$. $\mathcal{A}'$ simulates $\mathcal{O}_{\text{step}}$ and provides $\mathcal{A}$ with the next step of tags $T'_{i,j}$.

– To update the state of T_c in the challenge phase, $\mathcal{A}'$ proceeds as follows:
  - During the first iteration:
    1. $\mathcal{A}'$ computes two states. He computes $s^1_{T_c,1} = (c^1_{\text{ID}}, c^1_{\sigma}, c^1_{\phi_1})$ as if T_c will go through v_1 in the first iteration. He computes $s^1_{T_c,2} = (c^1_{\text{ID}}, c^1_{\sigma}, c^1_{\phi_2})$ as if T_c will go through v'_1 in the first iteration.
    2. $\mathcal{A}'$ then transmits $c^1_{\phi_1}$ and $c^1_{\phi_2}$ to oracle $\mathcal{O}_{\text{re-encrypt}}$.
    3. $\mathcal{O}_{\text{re-encrypt}}$ returns the result c'_{ϕ_b} of re-encrypting one of the two ciphertexts $c^1_{\phi_1}$, $c^1_{\phi_2}$ to $\mathcal{A}'$.
    4. $\mathcal{A}'$ writes the state $s^1_{T_c} = (c^1_{\text{ID}}, c^1_{\sigma}, c'_{\phi_b})$ into T_c.
  - In the next iterations, $\mathcal{A}'$ updates the state of T_c as if T_c will go through the sub-path $\overrightarrow{v'_2 v'_3 \ldots v'_\rho}$.

– At the end of the challenge phase, $\mathcal{A}$ reads the state of tag T_c and outputs b.

Note that the path stored in T_c is now either $\mathcal{P}_{T_c} = \overrightarrow{v_0 v_1 v'_2 \ldots v'_\rho}$ or $\mathcal{P}'_{T_c} = \mathcal{P}' = \overrightarrow{v_0 v'_1 v'_2 \ldots v'_\rho}$.

If $\mathcal{A}$ outputs $b = 1$, this means that T_c and T have a step in common that is different from v_0. Since $\mathcal{P}_{T_c} \cap \mathcal{P} = \{v_0, v_1\}$ and $\mathcal{P}'_{T_c} \cap \mathcal{P} = \{v_0\}$, outputting 1 implies that T_c went through $\mathcal{P}_{T_c}$ and hence through v_1. Therefore, the state that T_c stored at the first iteration corresponds to $\overrightarrow{v_0 v_1}$, and c'_{ϕ_b} is a re-encryption of $c^1_{\phi_1}$. $\mathcal{A}'$ outputs 1.

If $\mathcal{A}$ outputs $b = 2$, this means that T_c and T do not have a step in common except for v_0. This implies as well that T_c went through $\mathcal{P}'_{T_c} = \mathcal{P}'$ and hence through v'_1. Therefore, the state that T_c stored at the first iteration corresponds to $\overrightarrow{v_0 v'_1}$, and c'_{ϕ_b} is a re-encryption of $c^1_{\phi_2}$. $\mathcal{A}'$ outputs 2.

Therefore, if $\mathcal{A}$ has a non-negligible advantage ϵ in breaking TRACKER, $\mathcal{A}'$ as well has non-negligible advantage ϵ in breaking the semantic security of Elgamal under re-encryption, leading to a contradiction.

6 Evaluation

TRACKER can be implemented using today's available RFID tags. It requires tags to only store data, i.e, the encrypted ID, the encrypted HMAC and the encrypted path mark. Consequently, the tag stores three Elgamal ciphertexts $c_{\text{ID}} = (r_{\text{ID}} \cdot P, \text{ID} + r_{\text{ID}} \cdot Y)$, $c_\sigma = (r_\sigma \cdot P, \mathcal{M}(\text{HMAC}_k(\text{ID})) + r_{\text{ID}} \cdot Y)$ and $c_\phi = (r_\phi \cdot P, \mathcal{M}(\phi_{\text{ID}}(\mathcal{P}_{\text{valid}})) + r_\phi \cdot Y)$, which results in an overall storage of $2 \cdot 3 \cdot 160 = 960$ bits. Storing only 1 Kbit of data is feasible for today's EPC Class 1 Gen 2 UHF tags, for example Alien Technology's Higgs 3 tags [1].

Complexity for readers is also low in TRACKER. A reader R_i at step v_i is required to store an element $a_i \in \mathbb{F}_q$ and the public key of Elgamal pk. So, the total storage per reader is less than 80 bytes. Regarding computation, R_i is required to update the path mark of the tags passing by and to re-encrypt three ciphertexts: this sums up to a total three elliptic curve Elgamal encryptions. We conjecture this to be feasible even for lightweight embedded readers.

The manager M is the entity verifying the path that a tag T went through. Therefore, M is required to decrypt the ciphertexts stored on the tag using the secret key sk. M maintains two hash tables: the first table stores the list of valid paths in the supply chain. The second table is DB_{clone}. This is a hash table containing the IDs that M has read. So, the storage required for M is linear in the number of valid paths, and the number of tags in the supply chain $O(\nu + n)$, the path verification cost has constant complexity: when M reads a tag T, M is required to decrypt three elliptic curve ciphertexts to get ID, $\mathcal{M}(\text{HMAC}_k(\text{ID}))$ and $\mathcal{M}(\phi_{\text{ID}}(\mathcal{P}))$. Therewith, he computes a single HMAC and compares the output.

To detect cloning, M checks whether DB_{clone} contains ID. This operation is a hash look-up operation of cost $O(1)$. If no cloning is detected, M uses $\mathcal{M}(\phi_{\text{ID}}(\mathcal{P}))$ and $\text{HMAC}_k(\text{ID})$ to derive $\mathcal{M}(\phi(\mathcal{P}))$. Finally, M traces the tag path by looking up $\mathcal{M}(\phi(\mathcal{P}))$ into the table of valid paths.

In total, M performs three elliptic curve Elgamal decryptions, one HMAC verification, and two hash look-up operations per tag verification which is "cheap". As a conclusion, the complexity of TRACKER on the manager side is $O(n + \nu)$ storage and $O(1)$ computation.

Assume the size of an ID is, e.g., 96 bit as specified for EPC Class 1 Generation 2 tags, and each entry $\mathcal{M}(\phi(\mathcal{P}_{\text{valid}}))$ is 160 bit. A large sample TRACKER system supporting $n = 10^9$ different tags and $\nu = 10^6$ different valid paths would consume only around ≈ 11 GByte of storage for manager M. We conjecture this storage to be available for the manager of such a supply chain.

7 Related Work

Although historically one of the major applications for RFID tags, secure and privacy-preserving supply chain management has not received much attention in research. Instead, research focuses more on privacy-preserving authentication protocols and their cryptographic primitives [3].

Ouafi and Vaudenay [20] address counterfeiting of products using strong cryptography on RFID tags. To protect against malicious state updates, tags authenticate readers at every step in the supply chain. Only if readers are successfully authenticated, tags will update their internal state. Ouafi and Vaudenay [20] require tags to evaluate a cryptographic hash function twice: for reader authentication and for the state update. A similar approach with tags evaluating cryptographic hash functions is proposed by Li and Ding [17]. While such setups using cryptography-enabled tags might lead to a secure and privacy-preserving solution of the counterfeiting problem, tags will always be more expensive than read/write-only tags in TRACKER.

Chawla et al. [7] check whether covert channels exist in a supply chain that leak information about a supply chain's internal details to an adversary. Therefore, tags' state is frequently synchronized with a backend-database. If a tag's state contains "extra" data not in the database, the tag is rejected. TRACKER's focus, however, is on the secure, privacy-preserving detection of which path a tag has taken.

Shuihua and Chu [23] detect malicious tampering of a tag's state in a supply chain using watermarks. However, there is neither a way to identify a tag's path, nor to protect its privacy in the supply chain.

Kerschbaum and Oertel [15] detect counterfeits in the supply chain using pattern matching for anomaly detection. When a tag is read, this information is stored in a central database along with the ID of the tag. Unlike TRACKER, the focus of this paper is privacy-preservation of readers participating in the supply chain. There is no privacy for the tags in the supply chain.

Regarding simple product genuineness verification, solutions exist that rely on physical properties of a "tag". For example, TAGSYS produces holographic "tags" that are expensive to clone [24]. Verayo produces tags with Physically Unclonable Functions (PUF) [26]. While these approaches solve product genuineness verification, they neither support identification of tag's paths nor any kind of privacy properties.

Our construction based on polynomial path encoding might resemble other (cryptographic) constructions based on, e.g., Rabin fingerprints [21], aggregated messages authentication codes [14] or any kind of aggregated signatures. However, we stress that our design focuses on 1.) preserving both the order or sequence of steps in the supply chain and the privacy of paths and tags, 2.) at the same time putting only minimal computational burden on the manager ($O(1)$ complexity with low overhead), and 3.) being provable. While alternative constructions might be envisioned, this is far from being straightforward.

8 Conclusion

In this paper, we presented TRACKER to address security and privacy challenges in RFID-based supply chain management. TRACKER's main idea is to encode valid paths in a supply chain using polynomials. Readers representing steps in the supply chain evaluate polynomials successively, such that eventually the manager of the supply chain can uniquely identify the exact path a tag has taken. TRACKER's security, privacy, and unlinkability properties against adversaries relies on the semantic security of Elgamal and the security of HMAC, and we prove these properties. Contrary to related work, TRACKER does not require any computational complexity on the tag, but only 80 bytes of storage. This shows TRACKER's feasibility for today's cheap EPC Class 1 Gen 2 RFID tags.

References

[1] Alien Technology. RFID Tags, 2009. http://www.alientechnology.com/tags/index.php.

[2] 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.

[3] G. Avoine. RFID Security & Privacy Lounge, 2010. http://www.avoine.net/rfid/.

[4] M. Bellare. New Proofs for NMAC and HMAC: Security without Collision-Resistance. In *Proceedings of Annual International Cryptology Conference*, pages 602–619, Santa Barbara, USA, 2006. ISBN 3-540-37432-9.

[5] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proceedings of Annual International Cryptology Conference*, pages 1–15, Santa Barbara, USA, 1996. ISBN 3-540-61512-1.

[6] K. Brooks. Anti-Counterfeiting Initiatives and RFID Practices. *Contract Pharma*, Feb 2006. http://tinyurl.com/yj5pxct.

[7] K. Chawla, G. Robins, and W. Weimer. On Mitigating Covert Channels in RFID-Enabled Supply Chains. In *RFIDSec Asia*, Singapore, 2010. http://rfidsec2010.i2r.a-star.edu.sg.

[8] T. Dimitrou. rfidDOT: RFID delegation and ownership transfer made simple. In *Proceedings of International Conference on Security and privacy in Communication Networks*, Istanbul, Turkey, 2008. ISBN 978-1-60558-241-2.

[9] EU project SToP. Stop Tampering of Products, 2010. http://www.stop-project.eu/.

[10] 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.

[11] ICC Commercial Crime Services. Counterfeiting Intelligence Bureau, 2010. http://www.icc-ccs.org/index.php?option=com_content&view=article&id=29&Itemid=39.

[12] International Medical Products Anti-Counterfeiting Taskforce. International Medical Products Anti-Counterfeiting Taskforce – IMPACT, 2010. http://www.who.int/impact/.

[13] 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.

[14] J. Katz and A. Y. Lindell. Aggregate message authentication codes. In *Topics in Cryptology CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 155–169. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-79262-8.

[15] F. Kerschbaum and N. Oertel. Privacy-Preserving Pattern Matching for Anomaly Detection in RFID Anti-Counterfeiting. In *Workshop on RFID Security – RFIDSec'10*, Istanbul, Turkey, June 2010.

[16] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication, 1997. RFC 2104, http://www.ietf.org/rfc/rfc2104.txt.

[17] Y. Li and X. Ding. Protecting RFID communications in supply chains. In *Proceedings of ACM Symposium on Information, Computer and Communications Security*, pages 234–241, Singapore, 2007. ISBN 1-59593-574-6.

[18] Motorola. Saudi Arabia's luxury retailer Jade Jewellery implements Motorola's RFID technology to improve inventory management and security, 2010. http://tinyurl.com/yg6wzjv.

[19] G. Noubir, K. Vijayan, and H. J. Nussbaumer. Signature-based method for run-time fault detection in communication protocols. *Computer Communications Journal*, 21(5):405–421, 1998. ISSN 0140-3664.

[20] K. Ouafi and S. Vaudenay. Pathchecker: an RFID Application for Tracing Products in Suply-Chains. In *Workshop on RFID Security – RFIDSec'09*, pages 1–14, Leuven, Belgium, 2009. http://www.cosic.esat.kuleuven.be/rfidsec09/Papers/pathchecker.pdf.

[21] M.O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology. Harvard University, Cambridge, Massachusetts, USA, 1981.

[22] 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.

[23] H. Shuihua and C.-H. Chu. Tamper Detection in RFID-Enabled Supply Chains Using Fragile Watermarking. In *Proceedings of IEEE RFID*, pages 111–117, Las Vegas, USA, 2008.

[24] TAGSYS RFID. RFID Luxury Goods Solutions, 2010. http://www.tagsysrfid.com/Markets/Industries/Luxury-Goods.

[25] S. Vaudenay. On Privacy Models for RFID. In *Proceedings of ASIACRYPT*, pages 68–87, Kuching, Malaysia, 2007. ISBN 978-3-540-76899-9.

[26] Verayo. Verayo Anti-Counterfeiting Solution, 2010. http://www.verayo.com/solution/anti-counterfeiting.html.