Tracker: Security and Privacy for RFID-based Supply Chains

Erik-Oliver Blass Kaoutar Elkhiyaoui Refik Molva

EURECOM
2229 Route des Crêtes, BP 193
06560 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 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 [8, 9, 13, 17, 19]. 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 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 products. For example, World Health Organization (WHO) has estimated that 10% of U.S. pharmaceutical products were already counterfeit in 2005 [6]. 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.

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 on the other hand, 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. 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.
- Contrary to related work such as Ouafi and Vaudenay [15] or Li and Ding [12], 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

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

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 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 Vaudenay [18] and Ouafi and Vaudenay [15].

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 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. At the end, 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 path 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 $\mathcal{P}_{\mathrm{valid}_i}$ 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 $\{\mathcal{P}_{\mathrm{valid}_1}, \ldots, \mathcal{P}_{\mathrm{valid}_\nu}\}$, in a supply chain.

The last step v_l of a valid path $\mathcal{P}_{\mathrm{valid}_i} = \{v_0, \ldots, v_l\}$ 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.

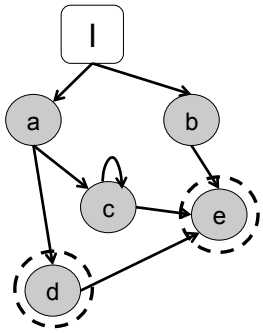

Figure 1: Simple supply chain. Checkpoints are encircled

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 con-

sists 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 $f_i : \mathcal{S} \rightarrow \mathcal{S}$
- 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 : $\mathcal{T} \rightarrow \mathcal{S}$ that reads out tag T_i and returns T_i's current state $s^j_{T_i}$
- a function WRITE: $\mathcal{T} \times \mathcal{S} \rightarrow \emptyset$ that writes a new state $s^{j+1}_{T_i}$ into tag T_i.
- a function CHECK: $\mathcal{S} \rightarrow \begin{cases} \mathcal{P}_{\text{valid}_i}, \text{ if tag } T_i \text{ went through } \mathcal{P}_{\text{valid}_i} \\ \emptyset, \text{ if } \nexists \mathcal{P}_{\text{valid}_i} \text{ that } T_i \text{ went through} \end{cases}$ that only based on a tag 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 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.

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 can be stated as follows: **if** the verification of tags 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 be always 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 [10].

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, cf., Algorithm 1, $\mathcal{A}$ can read from and write into arbitrary tags. 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.

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

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

Along these lines, the ITERATESUPPLYCHAIN command in Algorithm 1 enables $\mathcal{A}$ to iterate or "executes" the supply chain by one clock cycle, i.e., all tags advance by one step. $\mathcal{A}$ can execute the supply chain a total of ρ times. Now per iteration, per clock cycle, $\mathcal{A}$ can choose a set of r arbitrary tags, read-out their internal state, and re-write their state with some arbitrary data. Also, $\mathcal{A}$ has access to an "oracle" like construction QUERY$-\mathcal{O}_M$: queried with a tag $T_{i,j}$, QUERY$-\mathcal{O}_M$ 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. In summary, this definition is equivalent to the adversary proposed by Juels and Weis [10] and the *Non-Narrow Strong* adversary suggested by Vaudenay [18].

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

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

$\mathcal{A} \rightarrow M$: 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 it with $s^{j+1}_{T_i}$, or $\mathcal{A}$ can "create" its 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 *is said to be secure* $\Leftrightarrow$

For adversary $\mathcal{A}$, inequality

$$\Pr[\text{False Positive}] \leq \frac{|\mathcal{S}_{\text{valid}}|}{|\mathcal{S}|} + \epsilon$$

holds, where ϵ is negligible.

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 $\mathrm{DB}_{\mathrm{clone}}$. 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 $\mathrm{DB}_{\mathrm{clone}}$ – to check for cloning. Details about identifiers and handling of $\mathrm{DB}_{\mathrm{clone}}$ 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, as the cost of an active tag will not affect drastically the actual price of a luxury product, these products can be attached to tags with more computational capabilities that could implement access control and authentication to prevent cloning.

Furthermore, to get a malicious tag to be accepted by the manager, the adversary has to break-in the supply chain, clone a tag, inject its tag and get to the manager before the legitimate tag. We conjecture, this is not easy for an adversary to do, as the internal processes of the supply chain are well protected.

3.2 Privacy

An adversary $\mathcal{A}$ in TRACKER should not be able to tell if a tag T_i went through some step v in the supply chain based on the data stored on the tag.

While $\mathcal{A}$ can eavesdrop on communication between tags and readers over different protocol sessions or tamper with the data stored on the tags, it should not be able to violate tag privacy just by reading the data from the tag.

We illustrate this notion of privacy by a formal privacy experiment. In this experiment, $\mathcal{A}(r, s, \epsilon)$ has access to the tags in the supply chain in two phases. In the learning phase, $\mathcal{A}$ picks a step v in the supply chain through CHOOSESTEP(v), reads out and tampers with s tags that are going through the step v. It may as well read out any tag in the supply chain without exceeding r readings.

In the learning phase, $\mathcal{A}$ calls two types of oracle as shown in Algorithm 3. $\mathcal{O}_{\mathrm{pick}}$ is an oracle that randomly selects a tag from all the n tags in the supply chain.

$\mathcal{O}_{\mathrm{pick},v}$ is an oracle that returns a tag that went through the step v in the supply chain.

CHOOSESTEP(v);
for $i := 1$ **to** s **do**
  CHOOSETAG T_i = QUERY$-\mathcal{O}_{\mathrm{pick},v}$;
  s_{T_i} :=READ(T_i);
  WRITE(T_i, s'_{T_i});
end
for $j := 1$ **to** r **do**
  CHOOSETAG T'_j = QUERY$-\mathcal{O}_{\mathrm{pick}}$;
  $s_{T'_j}$:=READ(T'_j);
  WRITE($T'_j, s'_{T'_j}$);
end

Algorithm 3: Learning phase of adversary $\mathcal{A}$

In the challenge phase, $\mathcal{A}$ will be provided with an un-corrupted tag $T_{\text{challenge}}$,i.e., $\mathcal{A}$ did not write into $T_{\text{challenge}}$). Given the step v and the information $\mathcal{A}$ acquired during the learning phase, $\mathcal{A}$ outputs a bit b such that $b = 1$, if $T_{\text{challenge}}$ went through v and $b = 0$ otherwise. $\mathcal{A}$ is *s*uccessful if its guess is correct.

CHOOSETAG $T_{\text{challenge}}$ = QUERY$-\mathcal{O}_{\mathrm{pick}}$;
$s_{T_{\text{challenge}}}$:=READ($T_{\text{challenge}}$);
OUTPUT b;

Algorithm 4: Challenge phase of adversary $\mathcal{A}$

DEFINITION 3. TRACKER *is privacy preserving* $\Leftrightarrow$ *For adversary* $\mathcal{A}$*,*
$Pr(\mathcal{A}$ *outputs a right guess in the challenge phase*$) \leq \frac{1}{2} + \epsilon$ *where* ϵ *is negligible.*

3.3 Unlinkability

An adversary $\mathcal{A}$ can easily read the data stored on the tags. Therefore, TRACKER should prevent $\mathcal{A}$ from binding the data it reads to the tag. This differs from data privacy, a tag privacy could be met through encryption but not tag unlinkability – $\mathcal{A}$ may always be able to recognize the tag through the ciphertext it sends. Thus, the need to change the data sent by the tag regularly to prevent such a threat. In real world, tag unlinkability is the property that prevents an eavesdropper from tracking and distinguishing items and goods based on the non transient data they store – ID for instance.

Moreover, $\mathcal{A}$ may as well aim at distinguishing tags based on their paths. Unlike the ID, the path of the tag is ephemeral and it changes every time a tag T_i goes through a step v in the supply chain. Consequently, we need a new definition of unlinkability which is called "path unlinkability" that captures such property. Roughly speaking, path unlinkability should prevent an adversary $\mathcal{A}$ from telling if two tags T_i and T_j took the same path or not. In practice, path unlinkability will prevent an adversary $\mathcal{A}$ from binding a tag T_i to a palet of tags in the supply chain.

More formally, TRACKER should afford the following two types of unlinkability:

3.3.1 Path unlinkability

TRACKER should prevent an adversary $\mathcal{A}$ from being able to tell if a tag T_i went through the same path as a tag T_j that it has previously seen.

An adversary $\mathcal{A}(r, s, \epsilon)$ picks a tag T in the supply chain and it will be allowed to read out and write into up to s tags that went through the same path as T. Meanwhile, $\mathcal{A}$ can read out and write into up to r tags in the supply chain. The learning phase makes use of an additional oracle $\mathcal{O}_{\mathrm{pick},\mathcal{P}}$. $\mathcal{O}_{\mathrm{pick},\mathcal{P}}$ is an oracle that returns tags that went through path $\mathcal{P}$.

CHOOSETAG T = QUERY$-\mathcal{O}_{\mathrm{pick}}$;
s_T :=READ(T);
Let $\mathcal{P}$ denote the path T took;
for $i := 1$ **to** s **do**
  T_i = QUERY$-\mathcal{O}_{\mathrm{pick},\mathcal{P}}$;
  s_{T_i} :=READ(T_i);
  WRITE(T_i, s'_{T_i});
end
for $j := 1$ **to** r **do**
  CHOOSETAG T'_j = QUERY$-\mathcal{O}_{\mathrm{pick}}$;
  $s_{T'_j}$:=READ(T'_j);
  WRITE($T'_j, s'_{T'_j}$);
end

Algorithm 5: Learning phase of adversary $\mathcal{A}$

In the challenge phase, $\mathcal{A}$ is provided with a challenge tag $T_{\text{challenge}}$. Given the data stored on $T_{\text{challenge}}$, $\mathcal{A}$ outputs a bit b. $b = 1$ if $T_{\text{challenge}}$ went through the same path as T and $b = 0$ otherwise.

CHOOSETAG $T_{\text{challenge}} = \text{QUERY}-\mathcal{O}_{\text{pick}}$;
$s_{T_{\text{challenge}}} :=$READ$(T_{\text{challenge}})$;
OUTPUT b;

Algorithm 6: challenge phase of adversary $\mathcal{A}$

The adversary is *successful* if its guess is right.

DEFINITION 4. TRACKER *is said to provide path unlinkability* $\Leftrightarrow$ *For adversary* $\mathcal{A}$,
$Pr(\mathcal{A}$ *outputs a right guess in the challenge phase*$) \leq \frac{1}{2} + \epsilon$ *where* ϵ *is negligible.*

3.3.2 Tag unlinkability

As in Juels and Weis [10], in the learning phase, $\mathcal{A}(r, s, \epsilon)$ will be allowed to write into and read out up to r tags in the supply chain.

It will be as well provided with two challenge tags T_1 and T_2. $\mathcal{A}$ will have access to T_1 and T_2 at different steps of the supply chain and it will be allowed to read each tag up to s times.

CHOOSETAG $T_1 = \text{QUERY}-\mathcal{O}_{\text{pick}}$;
$s_{T_0} :=$READ(T_1^k);
CHOOSETAG $T_2 = \text{QUERY}-\mathcal{O}_{\text{pick}}$;
$s_{T_1} :=$READ$(T_2^{k'})$;
for $i := 1$ **to** s **do**
  ITERATESUPPLYCHAIN;
  $s_{T_1} :=$READ(T_1^{k+i});
  $s_{T_2} :=$READ$(T_2^{k'+i})$;
end
for $j := 2$ **to** $r + 2$ **do**
  CHOOSETAG $T_j = \text{QUERY}-\mathcal{O}_{\text{pick}}$;
  $s_{T_j} :=$READ(T_j);
  WRITE(T_j, s'_{T_j});
end

Algorithm 7: Learning phase of adversary $\mathcal{A}$

In the challenge phase, $\mathcal{A}$ is provided with a a tag T_b, $b \in \{1, 2\}$ through the oracle $\mathcal{O}_{\text{flip}}$. $\mathcal{O}_{\text{flip}}$ is an oracle that is provided with two tags T_1, T_2, randomly chooses $b \in \{1, 2\}$ and returns T_b.

Given the data stored on T_b and the result of the different readings $\mathcal{A}$ outputs its guess for the value of b.

CHOOSETAG $T_b = \text{QUERY}-\mathcal{O}_{\text{flip}}\{T_1, T_2\}$;
$s_{T_b} :=$READ(T_b);
OUTPUT b;

Algorithm 8: Challenge phase of adversary $\mathcal{A}$

$\mathcal{A}$ is successful if its guess of b is right.

DEFINITION 5. TRACKER *is said to provide tag unlinkability* $\Leftrightarrow$ *For adversary* $\mathcal{A}$,
$Pr(\mathcal{A}$ *outputs a right guess in the challenge phase*$) \leq \frac{1}{2} + \epsilon$ *where* ϵ *is negligible.*

4. TRACKER PROTOCOL

Overview: In TRACKER, a tag T's state s_T^l represents the sequence of steps in the supply chain 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, i.e., $s_T^l := Q_{\mathcal{P}_{\text{valid}}}(x_0)$. Now, TRACKER's security 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.

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 $|\mathcal{S}_{\text{valid}}|$ valid polynomials $Q_{\mathcal{P}_{\text{valid}_i}}(x_0)$. M checks whether one of this polynomials equals s_T^l, and if so, M knows the path the tag has taken.

Security and privacy: To protect security and privacy in TRACKER, tags store as state only encryptions of the polynomial path encoding, and readers use homomorphic (re-)encryption techniques for the arithmetic operations on encrypted state. At the end of the supply chain, the manager can then decrypt and identify the path.

Before the detailed protocol description in Section 4.3, the following paragraphs will first provide a quick overview about elliptic curve encryption used in this paper and TRACKER's polynomial path encoding.

4.1 Path Encoding in Tracker

TRACKER's polynomial path encoding is based on techniques for software fault detection. Noubir et al. [14] propose encoding 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. [14] and will be described in the following.

4.1.1 Polynomial path encoding

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, issuer step v_0 is associated with 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 $\mathcal{P} = \overrightarrow{v_0 v_1 \dots v_l}$ is defined in Equation 1:

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

(All operations are in $\mathbb{F}_q$.)

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 define the path *mark* as $\phi(\mathcal{P}) := Q_{\mathcal{P}}(x_0)$ and can therewith identify a path $\mathcal{P}$ using its polynomial evaluation $\phi(\mathcal{P})$.

Readers: These path marks are stored in the tag. A reader that is visited by a tag T, reads the T' current path mark, updates the path mark, and writes the updated path mark back into T. To eventually achieve the evaluation $\phi(\mathcal{P})$ of path $\mathcal{P} = \overrightarrow{v_0 v_1 \dots v_{i-1} v_i v_{i+1} \dots v_l}$, 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 $P_{i-1} = \overrightarrow{v_0 v_1 \dots v_{i-1}}$, and contains the path mark $\phi(P_{i-1})$.

To get $\phi(P_i)$, with $P_{i-1} = \overrightarrow{v_0 v_1 \dots v_i}$, reader R_i simply computes $\phi(P_i)$ using R_i's state transition function f_{R_i} defined as

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

So, $\phi(P_i) := f_{R_i}(\phi(P_{i-1})) = x_0 \cdot \phi(P_{i-i}) + a_i$. R_i stores $\phi(P_i)$ in T. By construction, this will eventually result in $\phi(\mathcal{P}) =$

$a_0 x_0^l + \sum_{i=1}^{l} a_i x_0^{l-i}$

The above path encoding with marks has the desired property that two different paths result in distinct path signatures with high probability. That is, $\forall \mathcal{P}, \mathcal{P}', \mathcal{P} \neq \mathcal{P}'$, equation $\phi(\mathcal{P}) = \phi(\mathcal{P}')$ holds with probability $\frac{1}{q}$, cf., Noubir et al. [14].

4.1.2 Tag state encoding and decoding

The state s_T^l of a valid tag T in the supply chain that went through a valid path $\mathcal{P}_{\text{valid}}$ consists of a tuple of three elements $s_T^l := (\text{ID}, \phi(\mathcal{P}_{\text{valid}}), \sigma(\mathcal{P}_{\text{valid}}, \text{ID}))$. The first element of that tuple is T's unique, random identifier $\text{ID} \in \mathbb{F}_q$. The second element is the mark of $\mathcal{P}_{\text{valid}}$ as given above.

The third element, $\sigma(\mathcal{P}_{\text{valid}}, \text{ID})$, acts as a *signature* combining ID and the path $\mathcal{P}_{\text{valid}}$ that T took. Details about σ will be given later in Section 4.3. The basic idea is that if $\mathcal{P}_{\text{valid}}$ along with ID are eventually identified by manager M, then M can use $\sigma(\mathcal{P}_{\text{valid}}, \text{ID})$ to verify whether it was really the tag with ID that actually went through $\mathcal{P}_{\text{valid}}$.

For decoding, M stores a set of $|\mathcal{S}_{\text{valid}}|$ valid marks $\phi(\mathcal{P}_{\text{valid}})$ and compares T's state to identify $\mathcal{P}_{\text{valid}}$. Then, M checks whether $\mathcal{P}_{\text{valid}}$ and ID match signature $\sigma(\mathcal{P}_{\text{valid}}, \text{ID})$.

As we will now see in the following paragraphs, tags in TRACKER store encrypted versions of ID and the path mark $\phi(\mathcal{P}_{\text{valid}})$. So in conclusion, a tag stores the tuple

$$s_T^l = (E(\text{ID}), E(\phi(\mathcal{P}_{\text{valid}})), \sigma(\mathcal{P}_{\text{valid}}, \text{ID})).$$

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 path mark $\phi(\mathcal{P})$ along with the encryption of its ID and signature $\sigma(\mathcal{P}_{\text{valid}}, \text{ID}))$. The use of Elgamal over elliptic curve requires a point mapping to transform some message $m \in \mathbb{F}_q$ to a point in the elliptic curve $\mathcal{E}$. We use two types of point mappings. One, for path mark ϕ to point mapping, is homomorphic, but not reversible. The other one, for ID to point mapping, is reversible, but not homomorphic.

4.2.1 Path mark to point mapping

To preserve the homomorphic property of Elgamal, we need a homomorphic mapping $\mathcal{M}_\phi : \mathbb{F}_q \to \mathcal{E}$ to map a mark $\phi(\mathcal{P})$ to a point in the elliptic curve such that $\forall m_1, m_2 \in \mathbb{F}_q, \mathcal{M}_\phi(m_1 + m_2) = \mathcal{M}_\phi(m_1) + \mathcal{M}_\phi(m_2)$.

We define our mapping of a mark $\phi(\mathcal{P}) \in \mathbb{F}_q$ to a point as $\mathcal{M}_\phi(\phi(\mathcal{P})) = \phi(\mathcal{P}) \cdot P \in \mathcal{E}$.

This mapping is a one-to-one mapping, but not reversible. This not an issue in TRACKER, as the manager knows the valid path marks in advance. Consequently, the manager computes and stores the *mapping* of the $|\mathcal{S}_{\text{valid}}|$ valid path marks, i.e., the $\mathcal{M}_\phi(\phi(\mathcal{P}_{\text{valid}_i})) \in \mathcal{E}$, instead of computing and storing the $|\mathcal{S}_{\text{valid}}|$ path marks $\phi(\mathcal{P}_{\text{valid}_i}) \in \mathbb{F}_q$.

4.2.2 ID to point mapping

Manager M has to be able to retrieve the ID of a tag T from T's state. The mapping of a tag's ID to a point in $\mathcal{E}$ calls for the use of any reversible mapping, e.g., such as the one introduced by Ateniese et al. [2]. In TRACKER, we use this mapping as a black box, and it will be denoted $\mathcal{M}$.

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.

4.3.1 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_0, $|k_0| = 160$ bit. The initial step v_0, representing the issuer in the supply chain, is associated with (a_0, k_0).

Similarly, I generates η random numbers $a_i \in \mathbb{F}_q, 1 \leq \eta$, and η random bit strings k_i, each of length $|k_i| = 160$ bit. I sends to each reader R_i, representing step v_i, the tuple (x_0, a_i, k_i) using a secure channel.

Also using a secure channel, I provides manager M with secret key sk, generator x_0, and tuples (i, a_i, k_i). Therewith, M is informed which reader R_i at step v_i knows which (a_i, k_i). As M knows $\mathcal{S}_{\text{valid}}$, i.e., which paths in the supply chain will be valid, he now computes all the $|\mathcal{S}_{\text{valid}}|$ valid path marks $\phi(\mathcal{P}_{\text{valid}})$ using Equation (1). Finally, M computes and stores pairs

$$(\mathcal{M}_\phi(\phi(\mathcal{P}_{\text{valid}_i})), \text{steps}),$$

where steps is the sequence of steps $\overrightarrow{v_0 v_{\mathcal{P}_{\text{valid}},1} v_{\mathcal{P}_{\text{valid}},2} \ldots v_{\mathcal{P}_{\text{valid}},l}}$ of $\mathcal{P}_{\text{valid}_i}$. That is, M knows for each mapping the sequence of steps.

In conclusion, x_0 is public, the (a_i, k_i) are secret and only known by reader R_i and M. M also knows sk.

4.3.2 Tag preparation

For each new tag T entering the supply chain, I draws a random identification $\text{ID} \in \mathbb{F}_q$ and two random numbers $r_\phi, r_{\text{ID}} \in \mathbb{F}_q$ to compute the following two ciphertexts:

$$\begin{aligned} c_{\text{ID}}^0 &= E(\text{ID}) = (U_{\text{ID}}, V_{\text{ID}}) = (r_{\text{ID}} \cdot P, \mathcal{M}(\text{ID}) + r_{\text{ID}} \cdot Y) \\ c_\phi^0 &= E(\phi(v_0)) = (U_\phi^0, V_\phi^0) = (r_\phi \cdot P, a_0 \cdot P + r_\phi \cdot Y) \end{aligned}$$

Now, let HMAC be a (secure) HMAC algorithm [5], $\text{HMAC}_k(m) : \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{F}_q$. Issuer I computes signature

$$\sigma^0(v_0, \text{ID}) := \text{HMAC}_{k_0}(\text{ID}).$$

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

4.3.3 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 $\mathcal{P} = \overrightarrow{v_0 v_1 \ldots v_{i-1}}$. R_i reads out T's current state $s_T^{i-1} = (c_{\text{ID}}^{i-1}, c_\phi^{i-1}, \sigma^{i-1})$.

Given the ciphertext $c_\phi^{i-1} = (U_\phi^{i-1}, V_\phi^{i-1})$, x_0 and a_i, R_i computes $c_\phi^i = (U_\phi^i, V_\phi^i)$:

$$
\begin{aligned}
U_\phi^i &= x_0 \cdot U_\phi^{i-1} = (x_0 r_\phi^{i-1}) \cdot P \\
V_\phi^i &= x_0 \cdot V_\phi^{i-1} + a_i \\
&= (a_0 x_0^i + \sum_{j=1}^{i} a_j x_0^{i-j}) \cdot P + (x_0 r_\phi^{i-1}) \cdot Y
\end{aligned}
$$

As you can see, the above is the homomorphic encryption variant of the reader computation of Section 4.1.1.

Then, using $\sigma^{i-1}(\mathrm{ID})$ stored in T, R_i computes

$$\sigma^i(\mathrm{ID}) = \mathrm{HMAC}_{k_i}(\sigma^{i-1}(\mathrm{ID})).$$

Reader R_i re-encrypts c_{ID}^{i-1}, c_ϕ^i. It picks randomly two numbers r'_{ID} and $r'_\phi \in \mathbb{F}_q$ and outputs two new ciphertexts

$$
\begin{aligned}
c_{\mathrm{ID}}^i &= (U_{\mathrm{ID}}^i, V_{\mathrm{ID}}^i) = (r'_{\mathrm{ID}} \cdot P + U_{\mathrm{ID}}^{i-1}, r'_{\mathrm{ID}} \cdot Y + V_{\mathrm{ID}}^i) \\
c_\phi'^i &= (U_\phi'^i, V_\phi^i) = (r'_\phi \cdot P + U_\phi^i, r'_\phi \cdot Y + V_\phi^i)
\end{aligned}
$$

Finally, R_i writes the new state $s_T^i = (c_{\mathrm{ID}}^i, c_\phi'^i, \sigma^i(\mathrm{ID}))$ into T.

4.3.4 *Path verification by M*

Manager M reads out a tag T's state $s_T^l = (c_{\mathrm{ID}}^l, c_\phi^l, \sigma^l(\mathrm{ID}))$.

First, M decrypts c_{ID}^l to get plaintext $\mathrm{ID} = D(c_{\mathrm{ID}}^l) \in \mathbb{F}_q$. M checks 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, decrypting c_ϕ^l will result in a point $\pi = D(c_\phi^l) = \phi(\mathcal{P}) \cdot P$. Now, M checks, whether π is in his list of valid mappings $\mathcal{M}_\phi(\phi(\mathcal{P}_{\mathrm{valid}_i}))$. If there is no match, M outputs $\emptyset$ and rejects the tag.

Finally, M checks T's signature. Without loss of generality, we assume that $\mathcal{M}_\phi(\phi(\mathcal{P}_{\mathrm{valid}_i}))$ leads M to the path $\mathcal{P}_{\mathrm{valid}} : \overrightarrow{v_0 v_1 \ldots v_l}$. Given $\sigma^l(\mathrm{ID})$ stored on the tag and secret keys $(k_0, k_1, \ldots k_l)$, M checks if the following equation holds:

$$\sigma^l(\mathrm{ID}) = \mathrm{HMAC}_{k_l}(\mathrm{HMAC}_{k_{l-1}}(\ldots(\mathrm{HMAC}_{k_0}(\mathrm{ID}))))$$

If it does, manager M outputs $\mathcal{P}_{\mathrm{valid}}$, adds ID to $\mathrm{DB}_{\mathrm{clone}}$, otherwise M outputs $\emptyset$ and rejects the tag.

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. [11].

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]:

Let $\mathcal{O}_{\mathrm{HMAC}_k}$ be an HMAC oracle that when it is provided with a message m, returns $\mathrm{HMAC}_k(m)$.

1. **Resistance to existential forgery:** An adversary $\mathcal{A}(N, \epsilon)$ can choose N messages $m_1, ..., m_N$, and provide them to the oracle $\mathcal{O}_{\mathrm{HMAC}_k}$ to get the corresponding $\mathrm{HMAC}_k(m_i)$. Still, the advantage ϵ of $\mathcal{A}(N, \epsilon)$ to 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:** even knowing m, $\mathcal{A}(N, \epsilon)$ cannot distinguish $\mathrm{HMAC}_k(m)$ from a random number. That is, HMAC_k is a pseudo-random function.

5.2 Security

THEOREM 1. *For $(\eta+1)$ randomly chosen keys $k_i, 0 \leq i \leq \eta$, if HMAC_{k_i} is resistant to existential forgery, then* TRACKER *is secure.*

The aim of $\mathcal{A}(\rho, r, \epsilon)$ is to win the security game, i.e., to come up with a tuple $(c'_{\mathrm{ID}}, c'_\phi, \sigma'_i)$ that will be accepted by the manager.

PROOF. Assume there would be an adversary $\mathcal{A}$ that breaks the security of TRACKER. That is, $\mathcal{A}(\rho, r, \epsilon)$ can provide a valid tuple $(c'_{\mathrm{ID}}, c'_\phi, \sigma'_i)$, then we construct an adversary $\mathcal{A}'(2\rho r, \epsilon)$ that breaks the resistance of existential forgery of HMAC_k.

$\mathcal{A}'(2\rho r, \epsilon)$ breaks the existential forgery of HMAC_k as follows:

- $\mathcal{A}'$ creates a TRACKER system with a supply chain of only one valid path $\mathcal{P}_{\mathrm{valid}} = \overrightarrow{v_0 v_1 \ldots v_l}$. It generates randomly the secret keys $k_0, k_1, ..., k_{l-1}$. $\forall 0 \leq i \leq l-1$, k_i is the key of the HMAC of step v_i. $\mathcal{A}'$ associates the step v_l with HMAC_k.

  $\mathcal{A}'$ generates a valid pair of keys (sk, pk) for Elgamal encryption.

- $\mathcal{A}'$ 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, $\mathcal{A}'$ provides $\mathcal{A}$ with r tags.

  1. $\mathcal{A}'$ picks r different randomly chosen $\mathrm{ID}_i \in \mathbb{F}_q$.
  2. Given its knowledge of the secret keys $k_i, 0 \leq i \leq l-1$ and the public key of Elgamal, $\mathcal{A}'$ can compute correctly HMAC_{k_i} corresponding to step v_i, the encryption of ID_i and the encryption of the path mark.
  3. To compute HMAC_k at step v_l, $\mathcal{A}'$ does the following:

     for *tags T_i of state $s_{T_i}^{l-1} := (c_{(\mathrm{ID},i)}^{l-1}, c_{(\phi,i)}^{l-1}, \sigma_i^{l-1})$ arriving at step v_l* **do**
     $\mathcal{A}' \xrightarrow{\sigma_i^{l-1}} \mathcal{O}_{\mathrm{HMAC}_k}$;
     $\sigma_i^l := \mathrm{HMAC}_k(\sigma_i^{l-1})$;
     $\mathcal{A}' \xleftarrow{\sigma_i^l} \mathcal{O}_{\mathrm{HMAC}_k}$;
     $s_{T_i}^l := (c_{(\mathrm{ID},i)}^l, c_{(\phi,i)}^l, \sigma_i^l)$;
     WRITE$(T_i, s_{T_i}^l)$;
     end

  4. $\mathcal{A}'$ gives the r tags T_i to $\mathcal{A}$.
  5. $\mathcal{A}'$ simulates the manager oracle $\mathcal{O}_{\mathrm{M}}$ as follows:

     for $i := 1$ *to* r **do**
       $s_{T_i}^l = (c_{(\mathrm{ID},i)}^l, c_{(\phi,i)}^l, \sigma_i^l)$;
       $\phi_i := D(c_{(\phi,i)}^l)$;
       if ϕ_i *is valid* **then**
         $\mathrm{ID}_i := D(c_{(\mathrm{ID},i)}^l)$;
         $\sigma_i^{l-1} := \mathrm{HMAC}_{k_{l-1}}(\ldots(\mathrm{HMAC}_{k_0}(\mathrm{ID}_i)))$;
         $\mathcal{A}' \xrightarrow{\sigma_i^{l-1}} \mathcal{O}_{\mathrm{HMAC}_k}$.
         $\sigma^l(\mathcal{P}_{\mathrm{valid}}, \mathrm{ID}_i) := \mathrm{HMAC}_k(\sigma_i^{l-1})$;
         $\mathcal{A}' \xleftarrow{\sigma^l(\mathcal{P}_{\mathrm{valid}}, \mathrm{ID}_i)} \mathcal{O}_{\mathrm{HMAC}_k}$
         if $\sigma_i^l := \sigma^l(\mathcal{P}_{\mathrm{valid}}, \mathrm{ID}_i)$ **then**
           OUTPUT 1;
         else
           OUTPUT 0;
       end
       else
         OUTPUT 0;
     end

- After the learning phase, $\mathcal{A}$ returns a new tuple $(c'_{\mathrm{ID}}, c'_{\phi}, \sigma'_i)$ to $\mathcal{A}'$.

Let ID' be the plaintext underlying the ciphertext c'_{ID}. $\mathcal{A}(\rho, r, \epsilon)$ breaks the security of TRACKER means that $\sigma'_i = \mathrm{HMAC}_k(\mathrm{HMAC}_{k_{l-1}}(\ldots(\mathrm{HMAC}_{k_0}(\mathrm{ID}'))))$.
Once $\mathcal{A}'$ receives $(c'_{\mathrm{ID}}, c'_{\phi}, \sigma'_i)$, it proceeds as follows:

1. It decrypts c'_{ID} and gets ID' using Elgamal secret key sk;
2. It provides the pair (m, σ'_i) where

$$m = \mathrm{HMAC}_{k_{l-1}}(\mathrm{HMAC}_{k_{l-2}}(\ldots(\mathrm{HMAC}_{k_0}(\mathrm{ID}'))))$$

Therefore, $\mathcal{A}'(2\rho r, \epsilon)$ breaks the existential forgery of HMAC_k in $N \leq 2\rho r$ queries with advantage ϵ. Therefore, the advantage of breaking the security of TRACKER is the same as the advantage of breaking HMAC.

Above, we have shown the equivalence between breaking an HMAC and TRACKER with one valid path. In the following, we show that if $\mathcal{A}(\rho, r, \epsilon)$ has an advantage ϵ to break the security of TRACKER with ν valid paths, $\mathcal{P}_{\mathrm{valid}_i}, 1 \leq i \leq \nu$, there would be an adversary $\mathcal{A}'(\rho, r, \epsilon')$ that breaks TRACKER with one valid path $\mathcal{P}_{\mathrm{valid}}$ and therefore, breaks HMAC's resistance to existential forgery with advantage $\epsilon' = \frac{\epsilon}{\nu}$.

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

5.3 Privacy Analysis

Let $k_0, ..., k_l$ be randomly chosen HMAC keys. Let a "cascaded" HMAC be defined as
$\mathrm{CHMAC}(m) := \mathrm{HMAC}_{k_l}(\mathrm{HMAC}_{k_{l-1}}(\ldots(\mathrm{HMAC}_{k_0}(m))))$.
The proofs for TRACKER's privacy and unlinkability make use of the following lemma:

LEMMA 1. **If** $\forall i, 0 \leq i \leq l, \mathrm{HMAC}_{k_i}$ *are pseudo-random functions,* **then** CHMAC *is as well a pseudo-random function.*

PROOF SKETCH. If we assume that there would be an adversary $\mathcal{A}_{\mathrm{distinguish}}$ (for short $\mathcal{A}$) that is able to distinguish the output of CHMAC from a random number, we can construct an $\mathcal{A}'_{\mathrm{distinguish}}$ (for short $\mathcal{A}'$) that distinguishes the output of HMAC_{k_l} on a message m from a random number.

Let $\mathcal{O}_{\mathrm{distinguish}}$ be the oracle query for the dinstinguishability game. Given the secret key k_l and a message m, it flips a coin $b \in \{0, 1\}$ and returns a message m'.

If $b = 0$, $\mathcal{O}_{\mathrm{distinguish}}$ returns a random number. If $b = 1$, $\mathcal{O}_{\mathrm{distinguish}}$ returns $\mathrm{HMAC}_{k_l}(m)$.

To break the indistinguishability property of HMAC_{k_l}, $\mathcal{A}'$ proceeds as following:

- $\mathcal{A}'$ generates l keys $k_i, 0 \leq i \leq l-1$.
- $\mathcal{A}'$ calls $\mathcal{A}$.
- $\mathcal{A}$ provides a message m.
- $\mathcal{A}'$ calls $\mathcal{O}_{\mathrm{distinguish}}$ and provides it with $\mathrm{HMAC}_{k_{l-1}}(\ldots \mathrm{HMAC}_{k_0}(m))$.
- $\mathcal{O}_{\mathrm{distinguish}}$ returns m' to $\mathcal{A}'$.
- $\mathcal{A}'$ provides $\mathcal{A}$ with m'.
- If $\mathcal{A}$ outputs 1 meaning that m' is $\mathrm{CHMAC}(m)$, $\mathcal{A}'$ outputs 1.
- If $\mathcal{A}$ outputs 0 meaning that m' is a random number, $\mathcal{A}'$ outputs 0.

Therefore, if $\mathcal{A}$ breaks the indistinguishability property of CHMAC with a non-negligible advantage, then $\mathcal{A}'$ breaks the indistinguishability property of HMAC_{k_l} with a non-negligible advantage. In conclusion, if HMAC_{k_l} is pseudo-random so is CHMAC. Also note that if $\forall 0 \leq i \leq l$, HMAC_{k_i} are pseudo-random, then is CHMAC as well. The output of CHMAC on a message m is indistinguishable from a random number. □

Consequently, given the indistinguishability property of HMAC and therewith the indistinguishability property of CHMAC, we reduce the proofs of privacy and the unlinkability of TRACKER to the semantic security of Elgamal.

Let $\mathcal{O}_{\mathrm{semantic}}$ be the oracle that, provided with two points $M_1, M_2 \in \mathcal{E}$, randomly chooses $b \in \{1, 2\}$, encrypts M_b using Elgamal and public key pk, and returns the resulting ciphertext $c_b = E(M_b)$.

5.3.1 Privacy

THEOREM 2. TRACKER *is privacy preserving under the DDH assumption and the security of* HMAC.

PROOF. Assume there would be an adversary $\mathcal{A}$ whose advantage in breaking TRACKER's privacy is not negligible. We construct a new adversary $\mathcal{A}'$ that executes $\mathcal{A}$ and breaks the semantic security of Elgamal. This leads to a contradiction under the DDH assumption.

$\mathcal{A}'$ breaks the semantic security of Elgamal as follows:

- Given the public key pk, $\mathcal{A}'$ specifies two different plaintexts m_1 and m_2.
- $\mathcal{A}'$ creates a TRACKER system with two step supply chain $\{v_1, v_2\}$ and an issuer at step v_0. It picks randomly an x_0 generator of $\mathbb{F}_q$. Then it selects a_0, a_1, a_2 such that the following equations hold: $m_1 = a_0 x_0 + a_1$ and $m_2 = a_0 x_0 + a_2$.

  Therefore, m_1 is the path mark corresponding to the path $\mathcal{P}_1 = \overrightarrow{v_0 v_1}$ and m_2 is the path mark corresponding to the path $\mathcal{P}_2 = \overrightarrow{v_0 v_2}$.

  Then, $\mathcal{A}'$ selects randomly k_0, k_1, k_2.
- $\mathcal{A}'$ starts $\mathcal{A}$ that goes into the learning phase:
  1. $\mathcal{A}$ picks a step $v_j, j \in \{1, 2\}$;
  2. $\mathcal{A}'$ simulates $\mathcal{O}_{\mathrm{pick}, v_j}$ and provides $\mathcal{A}$ with s tags $\{T_1, ..., T_s\}$ that went through v_j such that $(\mathrm{ID}_i, \phi(\mathcal{P}_j), \sigma^1(\mathcal{P}_j, \mathrm{ID}_i))$ is well constructed. Where ID_i is the identifier of tag T_i;
  3. $\mathcal{A}'$ provides $\mathcal{A}$ with additional r tags $\{T'_1, ..., T'_r\}$ simulating $\mathcal{O}_{\mathrm{pick}}$ such that the tuple $(\mathrm{ID}'_i, \phi(\mathcal{P}'_i), \sigma^1(\mathcal{P}'_i, \mathrm{ID}'_i))$ is well constructed. Where ID'_i is the identifier of tag T'_i and $\mathcal{P}'_i \in \{\mathcal{P}_1, \mathcal{P}_2\}$ is the path T'_i took.
- $\mathcal{A}'$ transmits $\{M_1 = m_1 \cdot P, M_2 = m_2 \cdot P\}$ to the challenge oracle $\mathcal{O}_{\mathrm{semantic}}$.

- Using the public key pk, $\mathcal{O}_{\text{semantic}}$ returns the encryption $c_b, b \in \{1,2\}$ of one of the points M_1, M_2 to $\mathcal{A}'$.
- $\mathcal{A}'$ prepares the challenge tag for $\mathcal{A}$:
  1. $\mathcal{A}'$ picks randomly ID $\in \mathbb{F}_q$, encrypts ID with the public key pk and gets c_{ID};
  2. it selects randomly $\sigma \in \mathbb{F}_q$.
  3. $\mathcal{A}$ writes c_b, c_{ID} and σ into a challenge tag $T_{\text{challenge}}$.
- $\mathcal{A}'$ calls the adversary $\mathcal{A}$ and provides it with the tag $T_{\text{challenge}}$, simulating $\mathcal{O}_{\text{pick}}$.
- Provided the indistinguishability property of lemma 1 of CHMAC, $\mathcal{A}'$ cannot distinguish whether σ is a random number or corresponds to an HMAC. Consequently, $\mathcal{A}'$ will simulate $\mathcal{O}_{\text{pick}}$ successfully.

As $\mathcal{A}$'s advantage in the privacy experiment is not negligible, $\mathcal{A}$ can decide if the tag $T_{\text{challenge}}$ went through the step v_j or not. Using this information $\mathcal{A}'$ can determine successfully which point M_b corresponds the the ciphertext c_b. Let assume $v_j = v_1$, if $\mathcal{A}$ guesses that $T_{\text{challenge}}$ went through v_1, this means that c_b corresponds to the encryption of M_1, otherwise it corresponds to the encryption of M_2. This breaks the semantic security of Elgamal that is ensured under the DDH assumption.

In the proof above, we have shown that breaking the privacy of TRACKER with two step supply chain and two paths is equivalent to breaking the semantic security of Elgamal . Now, we show that the privacy of TRACKER with η step supply chain and ν paths and the privacy of TRACKER with two step supply chain and two paths are equivalent.

As a point of fact, if there is an adversary $\mathcal{A}$ that breaks the privacy of TRACKER with η step supply chain and ν valid paths, we can construct an adversary $\mathcal{A}'$ that breaks the privacy of TRACKER with two step supply chain v_1, v_2 and two valid paths $\mathcal{P}_1 = \overrightarrow{v_0 v_1}$ and $\mathcal{P}_2 = \overrightarrow{v_0 v_2}$.

In order to do so, $\mathcal{A}'$ creates a TRACKER system with η step supply chain and ν valid paths as follows: the supply chain is a tree $\mathcal{T}$ of root v_0, that consists of of two subtrees $\mathcal{T}_1, \mathcal{T}_2$. The roots of $\mathcal{T}_1, \mathcal{T}_2$ are the steps v_1 and v_2 respectively. Therefore the steps of the supply chain belongs whether to $\mathcal{T}_1$ or $\mathcal{T}_2$. If $\mathcal{A}$ can tell if a tag T_i went through a step v_j in the supply chain, it can also tell which subtree $\mathcal{T}_k, k \in \{1,2\}$ T_i went through. Therefore, $\mathcal{A}'$ can tell if T_i went through v_1 or v_2. □

5.3.2 Path unlinkability

THEOREM 3. TRACKER *provides path unlinkability under the DDH assumption and the security of* HMAC.

PROOF. Assume there is an adversary $\mathcal{A}$ whose advantage in breaking the path unlinkability of TRACKER is not negligible. We therefore build a new adversary $\mathcal{A}'$ that executes $\mathcal{A}$ and breaks the semantic security of Elgamal.

$\mathcal{A}'$ breaks the semantic security of Elgamal as follows:

- $\mathcal{A}'$ specifies two plaintexts m_1 and m_2.
- $\mathcal{A}'$ creates a TRACKER system with two step supply chain $\{v_1, v_2\}$ and an issuer at step v_0. It picks a random generator of $\mathbb{F}_q$ x_0. It selects then a_0, a_1, a_2 such that the following equations hold: $m_1 = a_0 x_0 + a_1$ and $m_2 = a_0 x + a_2$.

  Therefore, m_1 is the path mark corresponding to the path $\mathcal{P}_1 = \overrightarrow{v_0 v_1}$ and m_2 is the path mark corresponding to the path $\mathcal{P}_2 = \overrightarrow{v_0 v_2}$.

  Then, $\mathcal{A}'$ selects randomly k_0, k_1, k_2.
- Given the public key pk, $\mathcal{A}'$ encrypts $m_1 \cdot P$ and writes the corresponding ciphertext c_1 into tag T.
- $\mathcal{A}'$ picks an ID ID $\in \mathbb{F}_q$ and encrypts it. Then, given the knowledge of the secret keys k_0, k_1, it computes $\sigma_1 = \sigma^1(\mathcal{P}_1, \text{ID})$. Finally, it writes the tuple $(c_{\text{ID}}, c_1, \sigma_1)$ into tag T.
- $\mathcal{A}'$ calls the adversary $\mathcal{A}$.
- Simulating $\mathcal{O}_{\text{pick}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with tag T.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\text{pick},\mathcal{P}_1}$ in the learning phase:

for $i := 1$ *to* s **do**
  PICK $\text{ID}_i \in \mathbb{F}_q$;
  $c_{(\text{ID},i)} := E(\text{ID}_i)$;
  $\sigma^1(\mathcal{P}_1, \text{ID}_i) := \text{HMAC}_{k_0}(\text{HMAC}_{k_1}(\text{ID}_i))$;
  $c_{(1,i)} :=$ RE-ENCRYPT(c_1);
  $s^1_{T_i} := (c_{(\text{ID},i)}, c_{(1,i)}, \sigma^1(\mathcal{P}_1, \text{ID}_i)$;
  WRITE$(T_i, s^1_{T_i})$
end

- Since $c_{(1,i)}$ is a re-encryption of c_1, it is also an encryption of $\phi(\mathcal{P}_1)$ and therefore, $\forall, 1 \leq i \leq s, T_i$ looks like a tag that went through the path $\mathcal{P}_1$.
- $\mathcal{A}'$ gives the tags to $\mathcal{A}$.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\text{pick}}$ and provides $\mathcal{A}$ with r arbitrary tags in the supply chain.
- $\mathcal{A}'$ transmits $M_1 = m_1 \cdot P$ and $M_2 = m_2 \cdot P$ to the challenge oracle $\mathcal{O}_{\text{semantic}}$.
- $\mathcal{O}_{\text{semantic}}$ returns the result c'_b of encrypting one of the two points M_1, M_2 to $\mathcal{A}'$.
- $\mathcal{A}'$ prepares the challenge tag $T_{\text{challenge}}$ for $\mathcal{A}$:
  1. $\mathcal{A}'$ picks an ID ID$' \in \mathbb{F}_q$;
  2. $\mathcal{A}'$ picks a random σ that stores onto the tag as the signature.
  3. $\mathcal{A}$ writes c'_b as the encryption of the path mark, along with the encryption of ID$'$ and σ into the challenge tag $T_{\text{challenge}}$. Finally, it provides $\mathcal{A}$ with the tag $T_{\text{challenge}}$.
- Provided the indistinguishability of the cascaded HMAC, $\mathcal{A}$ cannot distinguish σ from the valid signature. Therefore, $\mathcal{A}'$ will successfully simulate $\mathcal{O}_{\text{pick}}$.

Given that $\mathcal{A}$'s advantage in the path unlinkability experiment is not negligible, $\mathcal{A}$ can tell if the tag $T_{\text{challenge}}$ and the tag T_1 went through the same path with a non-negligible advantage. If $\mathcal{A}$ guesses that $T_{\text{challenge}}$ went through the same path as T_1, this means that c'_b is the encryption of M_1, otherwise c'_b is the encryption of M_2. This breaks the semantic security of Elgamal which is ensured under the DDH assumption.

Using the same approach as in the proof in section 5.3.1, we can show that the path unlinkability of TRACKER with η step supply chain and ν valid paths and the path unlinkability of TRACKER with two step supply chain and two valid paths are equivalent. □

5.3.3 Tag unlinkability

THEOREM 4. TRACKER *provides tag unlinkability under the DDH assumption and the security of* HMAC.

PROOF. Assume we have an adversary $\mathcal{A}$ whose advantage to break the unlinkability experiment is not negligible. We construct a new adversary $\mathcal{A}'$ that executes $\mathcal{A}$ and breaks the semantic security of Elgamal. To break the semantic security of Elgamal $\mathcal{A}'$ proceeds as follows:

- $\mathcal{A}'$ specifies two plaintexts m_1 and m_2.
- $\mathcal{A}'$ creates a supply chain for the TRACKER protocol.
- $\mathcal{A}'$ prepares the challenge tags T_1 and T_2 for $\mathcal{A}$:
  1. Given the public key pk, $\mathcal{A}'$ encrypts $\mathcal{M}(m_1)$ and $\mathcal{M}(m_2)$ and obtains the corresponding ciphertexts c_1, c_2 respectively. m_1 corresponds the the ID of tag T_1 and m_2 corresponds to the ID of tag T_2.
  2. $\mathcal{A}'$ picks a path $\mathcal{P}$ without loss of generality $\mathcal{P} = \overrightarrow{v_0 v_1 ... v_k}$.
  3. $\mathcal{A}'$ computes and encrypts $\phi(\mathcal{P})$ and gets $c_{(\phi,1)}$, then it computes $\sigma_1^k = \sigma^k(\mathcal{P}, m_1)$. It stores then $(c_1, c_{(\phi,1)}, \sigma_1^k)$ onto tag T_1.
  4. $\mathcal{A}'$ computes and encrypts $\phi(\mathcal{P})$ and gets $c_{(\phi,2)}$, then it computes $\sigma_2^k = \sigma^k(\mathcal{P}, m_2)$. It stores then $(c_2, c_{(\phi,2)}, \sigma_2^k)$ onto tag T_2.
  5. $\mathcal{A}'$ submits tags T_1 and T_2 to the adversary $\mathcal{A}$, simulating $\mathcal{O}_{\text{pick}}$. By construction T_1 and T_2 went through the same path.
  6. $\mathcal{A}'$ simulating $\mathcal{O}_{\text{pick}}$ provides $\mathcal{A}'$ with r additional tags.
  7. $\mathcal{A}'$ provides $\mathcal{A}$ with the data stored on T_1 and T_2 along s steps in the supply chain such that T_1 and T_2 keep on taking the same path.

     If T_1 and T_2 stores different encryption of the path mark, $\mathcal{A}'$ cannot tell if they went through the same path, given the path unlinkability proven above.
- $\mathcal{A}'$ transmits $M_1 = \mathcal{M}(m_1)$ and $M_2 = \mathcal{M}(m_2)$ to the challenge oracle $\mathcal{O}_{\text{semantic}}$.
- $\mathcal{O}_{\text{semantic}}$ returns the result c_b' of encrypting one of the two points M_1, M_2 and therewith m_1, m_2 to $\mathcal{A}'$.
- $\mathcal{A}'$ prepares the challenge tag $T_{\text{challenge}}$:
  1. $\mathcal{A}'$ picks a path $\mathcal{P}'$ such that $\mathcal{P}'$ is continuity of the path T_1 and T_2 took. If the last path mark that $\mathcal{A}$ reads from T_1 and T_2 corresponds to $\mathcal{P}_s = \overrightarrow{v_0 v_1 ... v_k ... v_{k+s}}$, $\mathcal{P}'$ should look like $\mathcal{P}' = \overrightarrow{\mathcal{P}_s ... v_j}$. Without loss of generality, $\mathcal{P}' = \overrightarrow{\mathcal{P}_s v_{k+s+1}}$ $\mathcal{A}'$ then, encrypts the corresponding path mark $\phi(\mathcal{P}')$ that it stores onto $T_{\text{challenge}}$.

     Since, T_1 and T_2 went through the same path by construction, $\mathcal{A}'$ does not have to know the value of b to provide a valid path mark that is compatible with what $\mathcal{A}$ has seen in the learning phase.
  2. $\mathcal{A}'$ selects randomly σ that it stores onto the tag $T_{\text{challenge}}$ as the signature.
  3. $\mathcal{A}'$ provides $\mathcal{A}$ with the tag $T_{\text{challenge}}$.
  4. As $\mathcal{A}$ cannot distinguish the output of cascaded HMAC and a random number, it cannot detect that $\mathcal{A}'$ is providing a non valid tuple. Therefore, $\mathcal{A}'$ simulates $\mathcal{O}_{\text{flip}}\{T_1, T_2\}$ successfully.

If $\mathcal{A}$'s advantage in the tag unlinkability experiment is not negligible, $\mathcal{A}$ can tell which tag $T_b, b \in \{1, 2\}$ corresponds to the challenge tag $T_{\text{challenge}}$. If it outputs $b = 1$, this means that $T_{\text{challenge}}$ corresponds to T_1 and therefore, it stores the encryption of the ID of T_1, i.e., $\mathcal{M}(m_1)$. Otherwise, it stores the encryption of $\mathcal{M}(m_2)$.

Therefore, $\mathcal{A}'$ can use $\mathcal{A}$ to break the semantic security of ELgamal that is ensured under the DDH assumption. □

6. EVALUATION

TRACKER requires tags to only store data, i.e, the encrypted ID, the encrypted path mark, and the signature. Consequently, the tag stores two Elgamal ciphertexts $c_{\text{ID}} = (r_{\text{ID}}P, \mathcal{M}(\text{ID}) + r_{\text{ID}}Y)$ and $c_\phi = (r_\phi P, \phi(\mathcal{P}_{\text{valid}})P + r_\phi Y)$, together with signature $\sigma(\mathcal{P}_{\text{valid}}, \text{ID})$. With a secure HMAC of output size of 160 bits, a tag stores $2 \cdot 2 \cdot 160 + 160 = 800$ bits. However, we can further optimize the storage on the tag by using the same Elgamal randomization factor for both ciphertexts. That is, $r_s = r_{\text{ID}}$. In this case, the tag stores $3 \cdot 160 + 160 = 640$ bits $= 80$ bytes. Storing only 80 bytes is feasible for today's EPC Class 1 Gen 2 UHF tags, for example Alien Technology's Higgs 3 tags [1].

In TRACKER a reader R_i at step v_i is required to store an element $a_i \in \mathbb{F}_q$, the public key of Elgamal pk, and an HMAC key k_i. So, the total storage per reader is 80 bytes. On the other hand, R_i is required to update the path mark of the tags passing by, to compute an HMAC, and to re-encrypt two ciphertexts, that is two Elgamal encryptions. We conjecture this is to be feasible even for 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 and to verify the signature using the secret keys k_i. M have two hash tables: the first table stores the list of valid paths in the supply chain and their corresponding HMAC keys. The second table is DB_{clone}. It is a hash table of the IDs that M has read. Whereas, the storage required on the manager side is linear in the number of valid paths $O(\nu)$ and the number of tags in the supply chain $O(n)$, the path verification cost is constant: when M reads a tag T, it decrypts the ciphertexts stored on T and gets ID and $\mathcal{M}_\phi(\phi(\mathcal{P}))$. To detect cloning, it checks if DB_{clone} contains ID. This operation is a look-up operation of cost $O(1)$. If no cloning is detected, M uses $\mathcal{M}_\phi(\phi(\mathcal{P}))$ to trace the tag path by looking up into the table of valid paths. And finally, if the path is valid, it verifies the signature stored on the tag against $\sigma(\mathcal{P}, \text{ID})$. M therefore, performs two decryptions, a signature verification and two look-up operations per tag. As a conclusion, the cost of TRACKER on the manager side is of $O(n + \nu)$ storage and $O(1)$ computation.

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 [15] 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 [15] 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 [12]. 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 [16] investigate the detection of 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.

Regarding simple product genuineness verification, solutions exits that rely on physical properties of a "tag". For example, TAGSYS produces holographic "tags" that are expensive to clone [17]. Verayo produces tags with Physically Unclonable Functions (PUF) [19]. While these approaches solve product genuineness verification, they do neither support identification of tag's paths, nor do they support any kind of privacy properties.

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 against adversaries relies on the semantic security of Elgamal and the seucrity of HMAC, and therefore, these properties are provable. 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] Giuseppe Ateniese, Jan Camenisch, and Breno 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] EU project SToP. Stop Tampering of Products, 2010. http://www.stop-project.eu/.

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

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

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

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