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, thereby 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 replicaprevention 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 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 database. 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 $s^i_{T_i}$. The set of all possible states is denoted with $S$, $s^j_{T_j} \in S$, and $|S|$ is a sufficiently large security parameter of TRACKER, e.g., $|S| = 2^{160}$.

- Issuer $I$: The issuer $I$ prepares tags for deployment. While attaching a tag $T_i$ to a product, $I$ writes an initial state $s^i_{T_i}$ 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 $s^i_{T_i}$ and writes an updated state $s^{i+1}_{T_i}$ into $T_i$. Here, $R_k$ uses some function $f_{R_k}$ to generate $s^{i+1}_{T_i}$ out of $s^i_{T_i}$, i.e., $f_{R_k}(s^i_{T_i}) = s^{i+1}_{T_i}$. 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 $s^i_{T_i}$ of $T_i$. Solely based on $s^i_{T_i}$, $M$ 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 $v$ in the supply chain is uniquely associated with a reader $R_i$.

Each directed edge $e \in E, e := v_i \overset{v_j}{\rightarrow}$, from vertex $v_i$ to vertex $v_j$, expresses that $v_i$ is a possible next step to step $v_j$ 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 $v_i \overset{v_j}{\rightarrow} \notin E$. 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 $R_i$ 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 $\forall l \in \{0, \ldots, l-1\} : v_l \overset{v_{l+1}}{\rightarrow} E$ and $l$ is the length of path $\mathcal{P}$. Clearly, different paths can have different path lengths.

A valid path $\mathcal{P}_{\text{valid}}$ 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 $\nu$ multiple different valid paths $\{\mathcal{P}_{\text{valid}0}, \ldots, \mathcal{P}_{\text{valid}\nu}\}$ in a supply chain.

The last step $v_l$ of a valid path $\mathcal{P}_{\text{valid}l} = \{v_0, \ldots, v_l\}$ 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 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 $\{I, a, c, c, c\}$. 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 $T$ of $n$ different tags
• a set of possible states $S$
• a total of $\eta$ different readers, $\eta = |E|$  
• issuer $I$ and manager $M$
• a set of $\eta$ state transition functions $f_i : S \rightarrow S$
• a set of $\nu$ valid paths
• a set of valid states $S_{\text{valid}}$
• a database $DB_{\text{clone}}$, stored at manager $M$ to protect against cloned tags (see next section)
• a function $\text{READ} : T \rightarrow S$ that reads out tag $T_i$ and returns $T_i$’s current state $s_{T_i}^j$
• a function $\text{WRITE}: T \times S \rightarrow \emptyset$ that writes a new state $s_{T_i}^{j+1}$ into tag $T_i$.
• a function $\text{CHECK} : S \rightarrow \{P_{\text{valid}}, \emptyset \}$, if tag $T_i$ went through $P_{\text{valid}}$, $\emptyset$, if $\emptyset P_{\text{valid}}$, that $T_i$ went through then only based on a tag $T_i$’s current state $s_{T_i}^j$, 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 $s_{T_i}^j$ by manager $M$ using CHECK returns a valid path $P_{\text{valid}}$, then $T_i$ must have gone through the steps of $P_{\text{valid}}$, 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 $P_{\text{valid}}$, 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 $A(\rho, r, \epsilon)$, or just $A$, has access to a Tracker system in two phases. First, in a learning phase, cf., Algorithm 1, $A$ 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
  \hspace{1cm} \text{ITERATESUPPLYCHAIN};
  \hspace{1cm} \text{for } j := 1$ to $r$ do
    \hspace{2cm} $\text{CHOOSETAG}(T_{i,j});$
    \hspace{2cm} $s_{T_{i,j}}^j := \text{READ}(T_{i,j});$
    \hspace{2cm} $\text{WRITE}(T_{i,j}, s_{T_{i,j}}^{j+1});$
    \hspace{2cm} $\text{QUERY} \leftarrow O_M(T_{i,j});$
  \hspace{1cm} end
end

Algorithm 1: Learning phase of adversary $A$

Along these lines, the \text{ITERATESUPPLYCHAIN} command in Algorithm 1 enables $A$ to iterate or “executes” the supply chain by one clock cycle, i.e., all tags advance by one step. $A$ can execute the supply chain a total of $\rho$ times. Now per iteration, per clock cycle, $A$ can choose a set of $r$ arbitrary tags, read-out their internal state, and re-write their state with some arbitrary data. Also, $A$ has access to an “oracle” like construction $\text{QUERY} \leftarrow O_M$: queried with a tag $T_{i,j}$, $\text{QUERY} \leftarrow O_M$ will return the output of the CHECK function.

The above definition of $A$ 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, $A$ enters the (simple) challenge phase as depicted in Algorithm 2:

$A \rightarrow M; T_i$

$M$ evaluates CHECK on $T_i$’s state;

Algorithm 2: Security challenge phase of adversary $A$

$A$ can either arbitrarily choose one tag $T_i \in T$, read and re-write it with $s_{T_{i}}^{j+1}$, or $A$ can “create” its own tag $T_i \notin T$ and write some state $s_{T_{i}}^{j+1}$ in it. Finally, $A$ 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 $\nu$ valid paths $P_{\text{valid}} = \{v_0, \ldots, v_l\}$, and if $T_i$ 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 $A$ 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 $\iff$

For adversary $A$, inequality

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

holds, where $\epsilon$ 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 $\text{DB}_{\text{clone}}$. 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 $\text{DB}_{\text{clone}}$ — to check for cloning. Details about identifiers and handling of $\text{DB}_{\text{clone}}$ 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 into 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 $A$ 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 $A$ 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, $A(r, s, \epsilon)$ has access to the tags in the supply chain in two phases. In the learning phase, $A$ picks a step $v$ in the supply chain through $\text{CHOOSESTEP}(v)$, 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, $A$ calls two types of oracle as shown in Algorithm 3. $O_{\text{pick}}$ is an oracle that randomly selects a tag from all the $n$ tags in the supply chain.

$O_{\text{pick}, s}$ is an oracle that returns a tag that went through the step $v$ in the supply chain.

$\text{CHOOSESTEP}(v)$;
for $i := 1$ to $s$ do
$\text{CHOOSETAG }T_i := \text{QUERY} - O_{\text{pick}, v}$;
$s_{T_i} := \text{READ}(T_i)$;
$\text{WRITE}(T_i, s_{T_i})$;
end
for $j := 1$ to $r$ do
$\text{CHOOSETAG }T_{ij} := \text{QUERY} - O_{\text{pick}}$;
$s_{T_{ij}} := \text{READ}(T_{ij})$;
$\text{WRITE}(T_{ij}, s_{T_{ij}})$;
end
Algorithm 3: Learning phase of adversary $A$

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

$\text{CHOOSETAG }T_{\text{challenge}} := \text{QUERY} - O_{\text{pick}}$;
$s_{T_{\text{challenge}}} := \text{READ}(T_{\text{challenge}})$;
OUTPUT $b$;
Algorithm 4: Challenge phase of adversary $A$

Definition 3. Tracker is privacy preserving $\iff$ For adversary $A$, $\Pr[A \text{ outputs a right guess in the challenge phase}] \leq \frac{1}{2} + \epsilon$ where $\epsilon$ is negligible.

3.3 Unlinkability
An adversary $A$ can easily read the data stored on the tags. Therefore, Tracker should prevent $A$ 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 — $A$ 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, $A$ 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 $A$ from telling if two tags $T_i$ and $T_j$ took the same path or not. In practice, path unlinkability will prevent an adversary $A$ from binding a tag $T_i$ to a palette 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 $A$ 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 $A(r, s, \epsilon)$ 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, $A$ can read out and write into up to $r$ tags in the supply chain. The learning phase makes use of an additional oracle $O_{\text{pick}, P}$. $O_{\text{pick}, P}$ is an oracle that returns tags that went through path $P$.

$\text{CHOOSETAG }T := \text{QUERY} - O_{\text{pick}}$;
$s_T := \text{READ}(T)$;
Let $P$ denote the path $T$ took;
for $i := 1$ to $s$ do
$T_i := \text{QUERY} - O_{\text{pick}, P}$;
$s_{T_i} := \text{READ}(T_i)$;
$\text{WRITE}(T_i, s_{T_i})$;
end
for $j := 1$ to $r$ do
$T_{ij} := \text{QUERY} - O_{\text{pick}}$;
$s_{T_{ij}} := \text{READ}(T_{ij})$;
$\text{WRITE}(T_{ij}, s_{T_{ij}})$;
end
Algorithm 5: Learning phase of adversary $A$

In the challenge phase, $A$ is provided with a challenge tag $T_{\text{challenge}}$. Given the data stored on $T_{\text{challenge}}$, $A$ outputs a bit $b$. $b = 1$ if $T_{\text{challenge}}$ went through the same path as $T$ and $b = 0$ otherwise.
CHOOSETAG $T_{\text{challenge}} = \text{QUERY} - \mathcal{O}_{\text{pick}}$;
$s_{T_{\text{challenge}}} := \text{READ}(T_{\text{challenge}})$;
OUTPUT $b$;

Algorithm 6: challenge phase of adversary $A$

The adversary is successful if its guess is right.

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

3.3.2 Tag unlinkability

As in Juels and Weis [10], in the learning phase, $A(r, s, \epsilon)$ 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 $T_2$. $A$ 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 $T_1 = \text{QUERY} - \mathcal{O}_{\text{pick}}$;
$s_{T_1} := \text{READ}(T_1)$;
CHOOSETAG $T_2 = \text{QUERY} - \mathcal{O}_{\text{pick}}$;
$s_{T_2} := \text{READ}(T_2)$;

Algorithm 7: Learning phase of adversary $A$

In the challenge phase, $A$ is provided with a tag $T_b$, $b \in \{1, 2\}$ through the oracle $\mathcal{O}_{\text{pick}}$. $\mathcal{O}_{\text{pick}}$ 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, $A$ outputs its guess for the value of $b$.

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

Algorithm 8: Challenge phase of adversary $A$

$A$ is successful if its guess of $b$ is right.

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

4. TRACKER PROTOCOL

**Overview:** In TRACKER, a tag $T$’s state $s_T$ 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 $P_{\text{valid}}$, a tag’s state $s_T$ will match the evaluation of a unique polynomial $Q_{P_{\text{valid}}}(x)$ in a fixed value $x_0$, i.e., $s_T := Q_{P_{\text{valid}}}(x_0)$. Now, TRACKER’s security relies on the property that for any two different paths $P \neq P'$, valid or not, the equation $Q_P(x_0) = Q_{P'}(x_0)$ 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 $s_I$ into a new tag $T$. 2.) Readers successively compute the evaluation of a polynomial: to achieve the evaluation of the “entire” polynomial $Q_{P_{\text{valid}}}(x_0)$ at the end of a valid path, each reader visited by tag $T$ computes $T$’s new state $s_T$ by applying simple arithmetic operations represented by the function $f_i$ on $T$’s current state $s_T^{-1}$. 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 $s_T$. $M$ knows a set of $\{s_{\mathcal{P}_{\text{valid}}} \}$ valid polynomials $Q_{P_{\text{valid}}}(x_0)$. $M$ checks whether one of these polynomials equals $s_T$, 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. Nouibir 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 Nouibir 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 $\mathcal{P} = v_0 v_1 \ldots v_\eta$ is defined in Equation 1:

\[
Q_\mathcal{P}(x) := a_0 x^0 + \sum_{i=1}^{\eta} a_i x^{i-1}.
\]

(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 $\mathbb{F}_q^\ast$. We define the path mark as $\phi(\mathcal{P}) := Q_\mathcal{P}(x_0)$ and can thereby 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 $\mathcal{P} = v_0 v_1 \ldots v_{i-1} v_i v_{i+1} \ldots v_\eta$, 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 $P_{i-1} = v_0 v_1 \ldots v_{i-1}$, and contains the path mark $\phi(P_{i-1})$.

To get $\phi(P_i)$, with $P_{i-1} = v_0 v_1 \ldots v_{i-1}$, 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, $\phi(P_i) := f_{R_i}(\phi(P_{i-1})) = x_0 \cdot \phi(P_{i-1}) + a_i$. $R_i$ stores $\phi(P_i)$ in $T$. By construction, this will eventually result in $\phi(\mathcal{P}) = \ldots$
4.1 Elliptic Curve Elgamal Cryptosystem

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

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

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

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

**Decryption:** To decrypt a ciphertext $c = (U, V)$, one computes $D(c) := U - sk \cdot V = M$.

In TRACER, a tag in the supply chain stores the elliptic curve Elgamal encryption of its path mark $\phi(P)$ 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 $E$. We use two types of point mappings. One, for path mark $\phi$ to point mapping, is homomorphic, but not reversible. The other one, for ID to point mapping, is reversible, but not homomorphic.

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 $E$ calls for the use of any reversible mapping, e.g., such as the one introduced by Atienese et al. [2]. In TRACER, we use this mapping as a black box, and it will be denoted $M$.

4.3 Detailed Protocol Description

TRACER 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 $\eta$ random numbers $a_i \in \mathbb{F}_q$, $1 \leq \eta$, and $\eta$ 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 $S_{\text{valid}}$, i.e., which paths in the supply chain will be valid, he now computes all the $|S_{\text{valid}}|$ valid path marks $\phi(\mathcal{P}_{\text{valid}})$ using Equation (1). Finally, $M$ computes and stores pairs

$$(M_0(\phi(\mathcal{P}_{\text{valid}})), \text{steps})$$

where steps is the sequence of steps $v_0 v_{\mathcal{P}_{\text{valid}} - 1} v_{\mathcal{P}_{\text{valid}} - 1} \ldots v_0$, of $\mathcal{P}_{\text{valid}}$. 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 ID $\in \mathbb{F}_q$ and two random numbers $r_{ID}, r_{ID-1} \in \mathbb{F}_q$ to compute the following two ciphertexts:

$c_{ID}^0 = E(\text{ID}) = (U_{ID}, V_{ID}) = (r_{ID} \cdot P, M(\text{ID}) + r_{ID} \cdot Y)$

$c_{ID}^0 = E(\phi(v_0)) = (U_0, V_0^0) = (r_{0} \cdot P, a_0 \cdot P + r_{0} \cdot Y)$

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

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

Finally, $I$ writes state $s_T^0 = (c_{ID}^0, c_{ID}^0, \sigma^0)$ 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 $P = v_i v_{i-1} \ldots v_1$. $R_i$ reads out $T$'s current state $s_T^{i-1} = (c_{ID}^{i-1}, c_{ID}^{i-1}, \sigma^{i-1})$.

Given the ciphertext $c_{ID}^{i-1} = (U_{ID}^{i-1}, V_{ID}^{i-1})$, $x_0$ and $a_i$, $R_i$ computes $c_{ID}^0 = (U_0^0, V_0^0)$;
5.2 Security

THEOREM 1. For \((\eta+1)\) randomly chosen keys \(k_i, 0 \leq i \leq \eta\), if \(\text{HMAC}_{k_i}\) is resistant to existential forgery, then \(\text{T}_{\text{RACKER}}\) is secure.

The aim of \(\mathcal{A}(\rho, r, \epsilon)\) is to win the security game, i.e., to come up with a tuple \((c'_{1D}, c'_{1}, \sigma')\) that will be accepted by the manager.

PROOF. Assume there would be an adversary \(\mathcal{A}\) that breaks the security of \(\text{T}_{\text{RACKER}}\). That is, \(\mathcal{A}(\rho, r, \epsilon)\) can provide a valid tuple \((c'_{1D}, c'_{1}, \sigma')\), then we construct an adversary \(\mathcal{A}'(2\rho, \epsilon)\) that breaks the resistance of existential forgery of \(\text{HMAC}_{k}\).

\(\mathcal{A}'(2\rho, \epsilon)\) breaks the existential forgery of \(\text{HMAC}_{k}\) as follows:

- \(\mathcal{A}'\) creates a \(\text{T}_{\text{RACKER}}\) with a supply chain of only one valid path \(P_{\text{valid}} = \overline{v_0}v_1 \ldots v_l\). It generates randomly the secret keys \(k_0, k_1, \ldots, k_{l-1}\). \(\forall 0 \leq i \leq l-1, k_i\) is the key of the HMAC of step \(v_i\). \(\mathcal{A}'\) associates the step \(v_i\) with \(\text{HMAC}_{k_i}\).
- \(\mathcal{A}'\) 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 \(\rho\) times. At each iteration of the supply chain, \(\mathcal{A}'\) provides \(\mathcal{A}\) with \(r\) tags.

1. \(\mathcal{A}'\) picks \(r\) different randomly chosen \(ID_i \in \mathbb{F}_q\).
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 \(\text{HMAC}_{k_i}\) corresponding to step \(v_i\), the encryption of \(ID_i\) and the encryption of the path mark.
3. To compute \(\text{HMAC}_{k}\) at step \(v_i\), \(\mathcal{A}'\) does the following:

   for tags \(T_i\) of state \(s'_{T_i} := (c'_{1D,i}, c'_{1,i}, \sigma'_{i-1})\) arriving at step \(v_i\) do
   
   \(A' \xrightarrow{s'_{T_i} \leftarrow O_{\text{HMAC}}};\)
   \(\sigma'_{i} := \text{HMAC}_{k_i}(\sigma'_{i-1});\)
   \(A' \xrightarrow{c'_{1D,i} \leftarrow O_{\text{HMAC}}};\)
   \(\sigma'_{1,i} := \text{HMAC}_{k_i}(c'_{1,i});\)
   \(A' \xrightarrow{\phi_{i} \leftarrow O_{\text{HMAC}}};\)
   \(\sigma'_{i} := \sigma'_{i}(P_{\text{valid}, ID_i});\)
   \(\text{WRITE}(T_i, s'_{T_i});\)
   
end

4. \(\mathcal{A}'\) gives the \(r\) tags \(T_i\) to \(\mathcal{A}\).
5. \(\mathcal{A}'\) simulates the manager oracle \(O_M\) as follows:

   for \(i := 1 \text{ to } r\) do
   
   \(s_{T_i} := (c'_{1D,i}, c'_{1,i}, \sigma'_i);\)
   \(\phi_{i} := D(c'_{1D,i});\)
   
   if \(\phi_{i} \text{ is valid then}\)
   
   \(ID_i := D(c'_{1D,i});\)
   \(\sigma'_{i-1} := \text{HMAC}_{k_{i-1}}(\ldots(\text{HMAC}_{k_0}(ID_i))));\)
   \(A' \xrightarrow{s_{T_i} \leftarrow O_{\text{HMAC}}};\)
   \(\sigma'_{i} := \sigma'_{i}(P_{\text{valid}, ID_i});\)
   \(\text{OUTPUT} 1;\)
   
else
   \(\text{OUTPUT} 0;\)
   
end

end
• After the learning phase, $\mathcal{A}$ returns a new tuple $(c'_{iD}, c'_b, \sigma'_i)$ to $\mathcal{A}'$.

Let ID' be the plaintext underlying the ciphertext $c'_{iD}$. $\mathcal{A}(\rho, \tau, \epsilon)$ breaks the security of TRAKER means that

$$\sigma'_i = \text{HMAC}_b(\text{HMAC}_{k_{l-1}}(\ldots (\text{HMAC}_{k_0}(\text{ID}'))))$$

Once $\mathcal{A}'$ receives $(c'_{iD}, c'_b, \sigma'_i)$, it proceeds as follows:

1. It decrypts $c'_{iD}$ and gets ID' using Elgamal secret key $sk$;

2. It provides the pair $(m, \sigma'_i)$ where

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

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

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

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

**5.3 Privacy Analysis**

Let $k_0, \ldots, k_l$ be randomly chosen HMAC keys. Let a “cascaded” HMAC be defined as

$$\text{CHMAC}(m) := \text{HMAC}_{k_{l}}(\text{HMAC}_{k_{l-1}}(\ldots (\text{HMAC}_{k_0}(m))))$$

The proofs for TRAKER’s privacy and unlinkability make use of the following lemma:

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

**Proof Sketch.** If we assume that there would be an adversary $\mathcal{A}_{\text{distinguish}}$ (for short $\mathcal{A}$) that is able to distinguish the output of CHMAC from a random number, we can construct an $\mathcal{A}'_{\text{distinguish}}$ (for short $\mathcal{A}'$) that distinguishes the output of $\text{HMAC}_{k_{l}}$ on a message $m$ from a random number.

Let $\mathcal{O}_{\text{distinguish}}$ be the oracle query for the distinguishment 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 $b = 0$, $\mathcal{O}_{\text{distinguish}}$ returns a random number. If $b = 1$, $\mathcal{O}_{\text{distinguish}}$ returns $\text{HMAC}_{k_{l}}(m)$.

To break the indistinguishability property of $\text{HMAC}_{k_{l}}$, $\mathcal{A}'$ proceeds as follows:

- $\mathcal{A}'$ 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 $\mathcal{O}_{\text{distinguish}}$ and provides it with $\text{HMAC}_{k_{l-1}}(\ldots (\text{HMAC}_{k_0}(m))$.
- $\mathcal{O}_{\text{distinguish}}$ returns $m'$ to $\mathcal{A}'$.

- $\mathcal{A}'$ provides $\mathcal{A}$ with $m'$.
- If $\mathcal{A}$ outputs 1 meaning that $m'$ is $\text{CHMAC}(m)$, $\mathcal{A}'$ 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 $\text{HMAC}_{k_{l}}$ with a non-negligible advantage. In conclusion, if $\text{HMAC}_{k_{l}}$ is pseudo-random so is CHMAC. Also note that if $0 \leq i \leq l$, $\text{HMAC}_{k_i}$ 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 TRAKER to the semantic security of Elgamal.

Let $\mathcal{O}_{\text{semantic}}$ be the oracle that, provided with two points $M_1, M_2 \in \mathbb{F}_p$, 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.** TRAKER 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 TRAKER’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 TRAKER 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 $\mathbb{F}_p$. Then it selects $a_0, a_1, a_2$ such that the following equations hold: $m_1 = a_0x_0 + a_1$ and $m_2 = a_0x_0 + a_2$.

Therefore, $m_1$ is the path mark corresponding to the path $P_1 = \overline{v_0v_1}$ and $m_2$ is the path mark corresponding to the path $P_2 = \overline{v_0v_2}$.

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 $\mathcal{O}_{\text{pick}, v_j}$ and provides $\mathcal{A}$ with $s$ tags $\{T_1, \ldots, T_s\}$ that went through $v_j$ such that $(\text{ID}_{t_1}, \sigma(T))$ is well constructed. Where $\text{ID}_{t_1}$ is the identifier of tag $T_1$;
3. $\mathcal{A}'$ provides $\mathcal{A}$ with additional $r$ tags $\{T_1', \ldots, T_r\}$ simulating $\mathcal{O}_{\text{pick}}$ such that the tuple $(\text{ID}_{t_1'}, \sigma(T'))$ is well constructed. Where $\text{ID}_{t_1'}$ is the identifier of tag $T_1'$ and $T_1' \in \{P_1, P_2\}$ is the path $T_1'$ took.

- $\mathcal{A}'$ transmits $\{M_1 = m_1 \cdot P, M_2 = m_2 \cdot P\}$ to the challenge oracle $\mathcal{O}_{\text{semantic}}$. 

the supply chain belongs whether to $T$, subtree $T_1$ can construct an adversary $A$ with two step supply chain and $P$ with two step supply chain is equivalent. The roots of $T_1$ and $T_2$ are the steps $v_1$ and $v_2$ respectively. Therefore the steps of the supply chain belongs whether to $T_1$ or $T_2$. If $A$ can tell if a tag $T_i$ went through a step $v_i$ in the supply chain, it can also tell which subtree $T_k$, $k \in \{1, 2\}$ $T_i$ went through. Therefore, $A'$ 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 $A$ whose advantage in breaking the path unlinkability of Tracker is not negligible. We therefore build a new adversary $A'$ that executes $A$ and breaks the semantic security of Elgamal.

$A'$ breaks the semantic security of Elgamal as follows:

- $A'$ specifies two plaintexts $m_1$ and $m_2$.
- $A'$ 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 $\mathbb{F}_q$. 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 $P_1 = \{v_0, v_1\}$ and $m_2$ is the path mark corresponding to the path $P_2 = \{v_0, v_2\}$.

Then, $A'$ selects randomly $k_0, k_1, k_2$.

- Given the public key $pk$, $A'$ encrypts $m_1 \cdot P$ and writes the corresponding ciphertext $c_1$ into tag $T$.
- $A'$ 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 $\sigma_1 = \sigma^1(P_1, ID)$. Finally, it writes the tuple $(c_1, k_1, \sigma_1)$ into tag $T$.

$A'$ calls the adversary $A$.

Simulating $O_{pick}$, $A'$ provides $A$ with tag $T$.

$A'$ simulates $O_{pick}$, $A'$ provides $A$ with two step supply chain and two valid paths are equivalent.

As $A'$'s advantage in the privacy experiment is not negligible, $A$ can decide if the tag $T_{challenge}$ went through the step $v_1$ or not. Using this information $A$ can determine whether point $M_0$ corresponds to the the ciphertext $c_0$. Let assume $v_1 = v_1$, if $A$ guesses that $T_{challenge}$ went through $v_1$, this means that $c_0$ 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 $\eta$ step supply chain and $\nu$ 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 $A$ that breaks the privacy of Tracker with $\eta$ step supply chain and $\nu$ valid paths, we can construct an adversary $A'$ that breaks the privacy of Tracker with two step supply chain $v_1, v_2$ and two valid paths $P_1 = \{v_0, v_1\}$ and $P_2 = \{v_0, v_2\}$.

In order to do so, $A'$ creates a Tracker system with $\eta$ step supply chain and $\nu$ valid paths as follows: the supply chain is a tree $T$ of root $v_0$, that consists of of two subtrees $T_1, T_2$. The roots of $T_1, T_2$ are the steps $v_1$ and $v_2$ respectively. Therefore the steps of the supply chain belongs whether to $T_1$ or $T_2$. If $A$ can tell if a tag $T_i$ went through a step $v_i$ in the supply chain, it can also tell which subtree $T_k$, $k \in \{1, 2\}$ $T_i$ went through. Therefore, $A'$ 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 $A$ whose advantage in breaking the path unlinkability of Tracker is not negligible. We therefore build a new adversary $A'$ that executes $A$ and breaks the semantic security of Elgamal.

$A'$ breaks the semantic security of Elgamal as follows:

- $A'$ specifies two plaintexts $m_1$ and $m_2$.
- $A'$ 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 $\mathbb{F}_q$. 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 $P_1 = \{v_0, v_1\}$ and $m_2$ is the path mark corresponding to the path $P_2 = \{v_0, v_2\}$.

Then, $A'$ selects randomly $k_0, k_1, k_2$.

- Given the public key $pk$, $A'$ encrypts $m_1 \cdot P$ and writes the corresponding ciphertext $c_1$ into tag $T$.
- $A'$ 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 $\sigma_1 = \sigma^1(P_1, ID)$. Finally, it writes the tuple $(c_1, k_1, \sigma_1)$ into tag $T$.

$A'$ calls the adversary $A$.

Simulating $O_{pick}$, $A'$ provides $A$ with tag $T$.

$A'$ simulates $O_{pick}$, $A'$ provides $A$ 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 $A'$ whose advantage to break the unlinkability experiment is not negligible. We construct a new adversary $A''$ that executes $A'$ and breaks the semantic security of Elgamal. To break the semantic security of Elgamal $A'$ proceeds as follows:

- $A'$ specifies two plaintexts $m_1$ and $m_2$.
- $A'$ creates a supply chain for the Tracker protocol.
- $A'$ prepares the challenge tags $T_1$ and $T_2$ for $A$:
  1. Given the public key $pk$, $A'$ encrypts $M(m_1)$ and $M(m_2)$ and obtains the corresponding ciphertexts $c_1, c_2$ respectively. $m_1$ corresponds to the ID of tag $T_1$ and $m_2$ corresponds to the ID of tag $T_2$.
  2. $A'$ picks a path $P$ without loss of generality $P = v_0v_1...v_k$.
  3. $A'$ computes and encrypts $\phi(P)$ and gets $c_{\phi(1)}$, then it computes $\sigma_1^k = \sigma^k(P, m_1)$. It stores then $(c_1, c_{\phi(1)}, \sigma_1^k)$ onto tag $T_1$.
  4. $A'$ computes and encrypts $\phi(P)$ and gets $c_{\phi(2)}$, then it computes $\sigma_2^k = \sigma^k(P, m_2)$. It stores then $(c_2, c_{\phi(2)}, \sigma_2^k)$ onto tag $T_2$.
  5. $A'$ submits tags $T_1$ and $T_2$ to the adversary $A$, simulating $O_{pick}$. By construction $T_1$ and $T_2$ went through the same path.
  6. $A'$ simulating $O_{pick}$ provides $A'$ with $r$ additional tags.
  7. $A'$ provides $A$ 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 encryptions of the path mark, $A'$ cannot tell if they went through the same path, given the path unlinkability proven above.

- $A'$ transmits $M_1 = M(m_1)$ and $M_2 = M(m_2)$ to the challenge oracle $O_{semantic}$.
- $O_{semantic}$ returns the result $c_s$ of encrypting one of the two points $M_1, M_2$ and therewith $m_1, m_2$ to $A'$.
- $A'$ prepares the challenge tag $T_{challenge}$:
  1. $A'$ picks a path $P'$ such that $P'$ is continuity of the path $T_1$ and $T_2$ took. If the last path mark that $A$ reads from $T_1$ and $T_2$ corresponds to $P_a = v_0v_1...v_k...v_{k+s}$, $P'$ should look like $P' = P_a...v_j$. Without loss of generality, $P' = P_a...v_{k+s+1}$, $A'$ then, encrypts the corresponding path mark $\phi(P')$ that it stores onto $T_{challenge}$.

Since, $T_1$ and $T_2$ went through the same path by construction, $A'$ does not have to know the value of $b$ to provide a valid path mark that is compatible with what $A$ has seen in the learning phase.

- $A'$ selects randomly $\sigma$ that it stores onto the tag $T_{challenge}$ as the signature.
- $A'$ provides $A$ with the tag $T_{challenge}$.

4. As $A$ cannot distinguish the output of cascaded HMAC and a random number, it cannot detect that $A'$ is providing a non valid tuple. Therefore, $A'$ simulates $O_{flip}(T_1, T_2)$ successfully.

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

Therefore, $A'$ can use $A$ 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 $c_{ID} = (r_1P, M(ID) + r_1Y)$ and $c_{\phi} = (r_\phi P, \phi(P_{valid})P + r_\phi Y)$, together with signature $\sigma(P_{valid}, ID)$. With a secure HMAC of output size of 160 bits, a tag stores $2 \cdot 2 \cdot 160 + 160 = 800$ bytes. However, we can further optimize the storage on the tag by using the same Elgamal randomization factor for both ciphertexts. That is, $r_s = r_{ID}$. 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 $R_1$ at step $v_i$ is required to store an element $a_i \in F_q$, the public key of Elgamal $pk$, and an HMAC key $k_s$. So, the total storage per reader is 80 bytes. On the other hand, $R_1$ 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 $k_s$. $M$ has 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 $M_{\phi}(\phi(P))$. 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 $M_{\phi}(\phi(P))$ 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 $\sigma(P, ID)$. $M$ 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, tag’s 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 exist 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 security 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


