CHECKER: On-site Checking in RFID-based Supply Chains

Kaoutar Elkhiyaoui
EURECOM
2229, route des Cretes
06560 Sophia Antipolis, France
elkhiyao@eurecom.fr

Erik-Oliver Blass*
College of Computer and Information Science
Northeastern University
Boston, MA 02115
blass@ccs.neu.edu

Refik Molva
EURECOM
2229, route des Cretes
06560 Sophia Antipolis, France
molva@eurecom.fr

ABSTRACT

Counterfeit detection in RFID-based supply chains aims at preventing adversaries from injecting fake products that do not meet quality standards. This paper introduces CHECKER, a new protocol for counterfeit detection in RFID-based supply chains through on-site checking. While RFID-equipped products travel through the supply chain, RFID readers can verify product genuineness by checking the validity of the product's path. CHECKER uses a polynomial-based encoding to represent paths in the supply chain. Each tag T in CHECKER stores an IND-CCA encryption of T's identifier `ID` and a signature of `ID` using the polynomial encoding of T's path as secret key. CHECKER is provably secure and privacy preserving. An adversary can neither inject fake products into the supply chain nor trace products. Moreover, RFID tags in CHECKER can be cheap read/write only tags that do not perform any computation. Per tag, only 120 Bytes storage are required.

Categories and Subject Descriptors

H.m [**Information Systems**]: Miscellaneous

General Terms

Security

Keywords

Privacy, RFID, Supply chain management

1. INTRODUCTION

One important application of RFID tags is product tracking and counterfeit detection in supply chains. In such a context, RFID tags are attached to products to enable product tracking along different partners in the supply chain.

In this paper, we propose a solution for genuineness verification based on RFID tags that allows product tracking while protecting the privacy of tags and partners in the supply chain. The main idea is to verify the genuineness of a product by verifying the validity of the path (sequence of partners) that the product went through in the supply chain as suggested by Blass et al. [4].

However, the solution presented in [4] has two major drawbacks: **1.)** It requires a centralized, trusted party called *"manager"* to carry out the path verification; otherwise, the manager is able to inject fake products into the supply chain. **2.)** The verification can only be performed once the tags arrive at the manager, but not before. This limits the wide deployment of such a solution, especially in a context where partners do not trust each other and demand to be able to verify product genuineness in real-time "on-site".

Contrary to Blass et al. [4], the solution presented in this paper addresses on-site checking by enabling each reader in the supply chain to verify the validity of the path taken by the tag, instead of a global path verification performed by a trusted party that only takes place at the final stage of the supply chain. Though such a solution will allow a faster and a more practical counterfeit detection, it comes with new threats to supply chain security and privacy.

With respect to security, the readers have to be able to verify the genuineness of a product by only reading the tag attached to the product. However, we have to make sure that these readers can by no means succeed in injecting fake products in the supply chain.

Furthermore, a product tracking system must take into account privacy concerns. Any solution that aims at tracking and tracing products is inherently exposed to malicious attacks targeting sensitive information about internal details and strategic relationships in the supply chain. Another requirement is the unlinkability of tags so that a reader in the supply chain must not be able to trace or tell tags apart once they leave its site. Moreover, a reader must not be able to learn any information about the path stored in a tag which has not visited its site.

Also, a secure and privacy preserving RFID-based solution has to be lightweight to allow wide deployment. Ideally, it should be suited to the cheapest RFID tags, i.e., read/write only tags. These tags come only with some re-writable memory and cannot perform any computation, let alone cryptographic operations. Moreover, the path verification at the readers should not be computationally heavy to avoid overloading readers and, thus hindering supply chain performance.

This paper introduces CHECKER, a secure and privacy preserving protocol for on-site genuineness verification and product tracking in supply chains using RFID tags. CHECKER stores in each tag T the tag identifier `ID` along with a signature of `ID`. The main idea behind CHECKER is that the secret key used to sign `ID` is an encoding of the path that T went through, thanks to an original combination of path encoding and signature. By verifying the signature in the tag, each reader thus validates the path taken that far, and by signing the `ID` the reader updates the path encoding. To protect T's

*Work done while at EURECOM.

WiSec'12, April 16–18, 2012, Tucson, Arizona, USA.

privacy, we encrypt T's ID and ID's signature using elliptic curve Cramer-Shoup encryption [8].

To summarize, CHECKER's main contributions are:

- In contrast to [4], CHECKER does not require a trusted party to perform path verification. Instead, CHECKER allows each reader in the supply chain to individually verify on-site whether a tag went through a valid path or not.
- CHECKER relies only on read/write only tags that are cheap and thus could allow wide deployment of CHECKER. A tag T in CHECKER is not required to perform any computation. T is only required to store its state that will be updated by readers along the supply chain.
- CHECKER is provably secure: an adversary cannot forge new tags. That is, an adversary cannot forge or change a tag T's state to convince a reader in the supply chain that T went through a valid path.
- CHECKER is provably privacy preserving: only readers in the supply chain can verify the validity of paths that tags have taken in the supply chain. Furthermore, an adversary cannot trace or link tags' interactions in the supply chain.
- Finally, CHECKER overcomes some limitations of the formal security and the privacy definitions of [4].

2. BACKGROUND

We use terms and notations in accordance with the ones used by Ouafi and Vaudenay [18] and by Blass et al. [4].

In this paper, the supply chain consists of a set of valid paths: an ordered sequence of steps, i.e., partner sites, that genuine products are allowed to visit.

Now, each read/write only RFID tag is attached to a product and it stores a history of the path that the product has taken. As in [4], each step of the supply chain is equipped with an RFID reader. Each reader reads out the state of tags in its vicinity and checks whether these tags went through a valid path in the supply chain or not. Finally, the reader updates the state of tags accordingly.

2.1 Entities

CHECKER involves the following entities:

Tags T_i: Each tag is attached to a single product or item. Each tag T_i is equipped with a re-writable memory storing T_i's current "state" denoted $s^j_{T_i}$.

Issuer I: The issuer I initializes tags at the beginning of the supply chain. It attaches each tag T_i to a product and writes an initial state $s^0_{T_i}$ into T_i.

Readers R_k: Each reader is associated with a single step in the supply chain. A reader R_k interacts with tags T_i in its range. He reads T_i's current state $s^j_{T_i}$ and based on a set $\mathcal{K}^V_k = \{K^1_k, K^2_k, ..., K^{\nu_k}_k\}$ of ν_k verification keys decides whether T_i went through a valid path or not. Once the verification phase is finished, R_k writes an updated state $s^{j+1}_{T_i}$ into T_i.

2.2 Supply chain

A supply chain is modeled as a digraph $G = (V, E)$, where V is the set of vertexes and E is the set of edges. Each vertex $\mathtt{v}_k \in V$ is a step in the supply chain that is uniquely associated with a reader R_k. On the other hand, each edge $e \in E$, $e := \overrightarrow{\mathtt{v_i v_j}}$, denotes a valid transition from $\mathtt{v_i}$ to $\mathtt{v_j}$. The issuer I is represented in G as being the only vertex with indegree equals to 0 denoted $\mathtt{v}_0$.

A path in a supply chain P is a finite ordered set of steps $\mathtt{P} = \{\mathtt{v}_0, \mathtt{v}_1 \ldots, \mathtt{v}_l\}$, where $\forall i \in \{0, \ldots, l-1\} : \overrightarrow{\mathtt{v}_i\mathtt{v}_{i+1}} \in E$, and l is the length of path P.

Naturally, the supply chain contains a set of valid paths $\mathtt{P}_{\mathrm{valid}_i}$, which are the set of paths that genuine products are allowed to go through.

Contrary to [4], in this paper we do not assume the existence of a *"manager"* that checks the validity of the path that a product has undertaken. Instead, CHECKER attempts to allow each reader in the supply chain to verify whether the products that it is presented with went through a valid path or not.

2.3 A CHECKER System

A CHECKER system comprises the following:

- A supply chain $G = (V, E)$.
- A set $\mathcal{T}$ of n tags.
- A set of possible states $\mathcal{S}$ that could be stored into tags.
- A set $\mathcal{R}$ of η readers R_k.
- Each reader R_k knows a set $\mathcal{P}_k = \{\mathtt{P}^1_k, \mathtt{P}^2_k, ..., \mathtt{P}^{\nu_k}_k\}$ of ν_k valid paths leading to R_k.
- Also, reader R_k has a set $\mathcal{K}^k_V = \{K^1_k, K^2_k, ..., K^{\nu_k}_k\}$ of ν_k verification keys. Each verification key K^j_k corresponds to a valid path $\mathtt{P}^j_k$.
- Issuer I.
- A set of valid states $\mathcal{S}_{\mathrm{valid}}$. If tag T_i stores a state $s^j_{T_i} \in \mathcal{S}_{\mathrm{valid}}$, then this implies that T_i took a valid path in the supply chain with high probability.
- A function ITERATESUPPLYCHAIN: When called, tags advance by one step in the supply chain and they are read and re-written by readers.
- A function READ : $\mathcal{T} \to \mathcal{S}$ that reads tag T_i and outputs T_i's current state $s^j_{T_i}$.
- A function WRITE: $\mathcal{T} \times \mathcal{S} \to \mathcal{S}$ that writes a new state $s^{j+1}_{T_i}$ into tag T_i.
- A function CHECK: $\mathcal{R} \times \mathcal{T} \to \{0, 1\}$ performed by readers in the supply chain. Based on T_i's current state $s^j_{T_i}$ and the set of verification key $\mathcal{K}^V_k$ of a reader R_k, CHECK decides whether T_i went through a valid path in the supply chain that is leading to R_k or not.

$$\text{CHECK}(R_k, T_i) : \mathcal{S} \to \begin{cases} \text{1, if tag } T_i \text{ went through a} \\ \text{valid path } \mathtt{P}^j_k \in \mathcal{P}_k \\ \text{or 0, if } \nexists\, \mathtt{P}^j_k \in \mathcal{P}_k \text{ that matches} \\ T_i\text{'s state.} \end{cases}$$

3. ADVERSARY MODEL

Readers in CHECKER are supposed to read the state stored into tags, check whether the tags took a valid path in the supply chain and then update the tags' states accordingly. We assume that readers' corruption is possible. That is, readers can try tracking tags in order to spy on other readers, as well as injecting fake products in the supply chain.

Moreover, we assume that the issuer I in CHECKER is honest and cannot be corrupt by adversaries. This implies that when tags are initialized at the beginning of the supply chain by I, these tags will definitely meet the supply chain requirements and quality standards. However, these tags may later in the supply chain be corrupt by adversaries.

As CHECKER relies on read/write only tags to implement product tracking, an adversary $\mathcal{A}$ against CHECKER is not only allowed to eavesdrop on tags' communication but to also tamper with tags' internal state. $\mathcal{A}$ can as well have access to the communication between tags and readers and know the steps $\mathtt{v}_k$ that a tag T is visiting. He can also monitor a step $\mathtt{v}_k$ in the supply chain by eavesdropping on tags going into or leaving the step $\mathtt{v}_k$.

To capture these capabilities in our definitions, an adversary $\mathcal{A}$ has access to the following oracles:

- $\mathcal{O}_{\texttt{Draw}}(\texttt{condition})$: When queried with a condition $\mathtt{c}$, $\mathcal{O}_{\texttt{Draw}}$ randomly selects a tag T from the n tags $\mathcal{T}$ in the supply chain that satisfies the condition $\mathtt{c}$ and returns T to $\mathcal{A}$. For example:
  1. To have access to a tag T which just entered the supply chain, i.e., T is at step $\mathtt{v}_0$, $\mathcal{A}$ queries the oracle $\mathcal{O}_{\texttt{Draw}}$ with condition $\mathtt{c}$ = "tag at step $\mathtt{v}_0$".
  2. To have access to a tag T whose identifier is ID, $\mathcal{A}$ calls the oracle $\mathcal{O}_{\texttt{Draw}}$ with condition $\mathtt{c}$ = "tag with identifier ID". $\mathcal{O}_{\texttt{Draw}}$ returns a tag with identifier ID if there is any.
  3. To have access to a tag T whose next step in the supply chain is step $\mathtt{v}_k$, $\mathcal{A}$ queries the oracle $\mathcal{O}_{\texttt{Draw}}$ with condition $\mathtt{c}$ = "tag's next step is $\mathtt{v}_k$".

  We indicate that adversary $\mathcal{A}$ can query the oracle $\mathcal{O}_{\texttt{Draw}}$ with any combination of disjunctions or conjunctions of conditions.
- $\mathcal{O}_{\texttt{Check}}(R_k, T)$: On input of reader R_k and tag T, $\mathcal{O}_{\texttt{Check}}$ returns the output of the CHECK function performed by reader R_k for tag T.
- $\mathcal{O}_{\texttt{Step}}(T)$: On input of tag T, the oracle $\mathcal{O}_{\texttt{Step}}(T)$ returns the *next* step of tag T in the supply chain.
- $\mathcal{O}_{\texttt{Flip}}(T_0, T_1)$: On input of two tags T_0 and T_1, $\mathcal{O}_{\texttt{Flip}}$ flips a coin $b \in \{0, 1\}$ and returns tag T_b to $\mathcal{A}$.
- $\mathcal{O}_{\texttt{Corrupt}}(R_k)$: On input of reader R_k, the oracle $\mathcal{O}_{\texttt{Corrupt}}$ returns the secret information S_k associated with reader R_k to $\mathcal{A}$. We say that $\mathcal{A}$ corrupted the step $\mathtt{v_k}$ associated with reader R_k.

Note that whenever $\mathcal{A}$ is given access to a tag T, $\mathcal{A}$ is allowed to read from T by calling the function READ and to write into T through the function WRITE.

By having access to these oracles, an adversary $\mathcal{A}$ is able **1.)** to corrupt readers, **2.)** to have an arbitrary access to tags, and **3.)** to monitor readers in the supply chain.

3.1 Security

The security goal of CHECKER is to prevent an adversary $\mathcal{A}$ from forging a valid state for a tag T_i that did not go through a valid path in the supply chain. This goal matches the *soundness* property of the CHECK function.

More formally, **if** on input of a of tag T_i and reader R_k, the function CHECK(R_k, T_i) outputs 1, i.e., there is a path $\mathtt{P}_k^j \in \mathcal{P}_k$ that corresponds to the state s_{T_i} stored into T_i, **then** we conclude that T_i must have gone through $\mathtt{P}_k^j$ (with high probability).

It is important to note that when we say that a tag T_i went through path $\mathtt{P} = \overrightarrow{\mathtt{v}_0\mathtt{v}_1...\mathtt{v}_l}$, this means that tag T_i was issued by I and that the state of T_i has been updated correctly by using the secrets of readers $R_1, R_2, ..., R_l$ in that order. It does not mean that T_i went actually through the steps composing the path P. If we imagine a scenario where an adversary $\mathcal{A}$ knows all the readers' secrets, $\mathcal{A}$ can update the state of any tag T_i and make it look as if T_i went through some path P.

Now, we say that CHECKER is *sound*, if and only if, a reader R_k in the supply chain accepts a tag T_i only when the state of tag T_i has been updated correctly using the secrets of readers in some valid path leading to R_k.

We formalize soundness using an experiment-based definition as in [4]. In this experiment, an adversary $\mathcal{A}$ runs in two phases. First in the learning phase as depicted in Algorithm 1, $\mathcal{A}$ can corrupt up to r readers R_i of his choice by calling the oracle $\mathcal{O}_{\texttt{Corrupt}}$.

Then, $\mathcal{A}$ is allowed to iterate the supply chain up to ρ times by calling the function ITERATESUPPLYCHAIN. Whenever called, the function ITERATESUPPLYCHAIN advances the tags to their next step.

In each iteration of the supply chain, $\mathcal{A}$ can call the oracle $\mathcal{O}_{\texttt{Draw}}$ to get up to s tags $T_{(i,j)}$ that satisfy some condition $\mathtt{c}_{(i,j)}$ specified by $\mathcal{A}$. $\mathcal{A}$ can read from and write into these tags $T_{(i,j)}$. He can as well query the function CHECK for each tag $T_{(i,j)}$.

```
for i := 1 to r do
  | S_i ← O_Corrupt(R_i);
end
for i := 1 to ρ do
  | ITERATESUPPLYCHAIN;
  | for j := 1 to s do
  |   | T_(i,j) ← O_Draw(c_(i,j)) ;
  |   | s^i_T(i,j) :=READ(T_(i,j));
  |   | WRITE(T_(i,j), s'^i_T(i,j));
  |   | b_T(i,j) ← O_Check(R_T(i,j), T_(i,j));
  | end
end
```

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

```
T_c ← A;
for i := 1 to η do
  | b_(i,T_c) ← O_Check(R_i, T_c);
end
```

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

Finally in the challenge phase, $\mathcal{A}$ selects a challenge tag $\mathtt{T}_c$ that he gives to the oracle $\mathcal{O}_{\texttt{Check}}$, cf., Algorithm 2. $\mathcal{O}_{\texttt{Check}}$ outputs a set of η bits $b_{(i,\mathtt{T}_c)}$ such that $b_{(i,\mathtt{T}_c)} = $ CHECK$(R_i, \mathtt{T}_c)$.

$\mathcal{A}$ is said to be successful if and only if:

i.) $\exists$ R_i such that CHECK$(R_i, \mathtt{T}_c) = 1$, i.e., there is a path $\mathtt{P}_i^j$ that corresponds to $\mathtt{T}_c$'state; **ii.)** $\exists$ $\mathtt{v} \in \mathtt{P}_i^j$ such that step v is not corrupted by $\mathcal{A}$; **iii.)** and finally, $\mathtt{T}_c$ did not go through step v.

DEFINITION 1 (SECURITY). CHECKER *provides security* $\Leftrightarrow$ *For adversary* $\mathcal{A}$, *inequality* $Pr[\mathcal{A}$ *is successful* $] \leq \frac{|\mathcal{S}_{\text{valid}}|}{\mathcal{S}} + \epsilon$ *holds, where* ϵ *is negligible.*

The adversary $\mathcal{A}$ captured by the definition above is a non narrow strong adversary in the sense of [22]. He can access tags arbitrarily and tamper with their states. He is also allowed to access the output of the protocol and corrupt readers. In the real world, such an adversary corresponds to a partner in the supply chain whose goal is to inject fake products.

Note. As we use read/write only tags, *completeness* of CHECKER cannot be ensured. An adversary $\mathcal{A}$ can always tamper with tags' internal states by writing dummy data into them. Thus, $\mathcal{A}$ can always invalidate the state of T_i leading the CHECK function to output 0.

Cloning. CHECKER targets read/write only tags to perform on-site checking. As a result, any malicious entity can read and rewrite the content of tags, and therewith, it can clone tags. Such an attack cannot be prevented in a setting that relies on read/write only tags which cannot implement any reader authentication.

To mitigate this problem, each partner P_i in the supply chain keeps a database $\texttt{DB}_i$ that contains the identifiers of tags present at P_i's site. Then, time is divided into epochs e_k (typically, the duration of an epoch e_k is one day) and partners are required to update their databases at the beginning of each epoch e_k.

To detect clones, each pair of partners P_i and P_j invoke a protocol for privacy preserving set intersection [9, 10] at the beginning of each epoch e_k, to check whether there is an identifier ID that is present in both of their databases. At the end of the privacy preserving set intersection protocol, both partners obtain a set of identifiers $S_{(i,j)} = \texttt{DB}_i \cap \texttt{DB}_j$ that represent the clones in their sites. If $S_{(i,j)} \neq \emptyset$, then P_i and P_j can discard the clones and investigate where the clones come from.

3.2 Privacy

In line with previous work [4], a privacy preserving verification of product genuineness in the supply chain should meet the two following requirements:

1.) An adversary $\mathcal{A}$ must not be able to distinguish between tags based on their interactions with the readers in the supply chain or based on their interactions with $\mathcal{A}$. This requirement deals with tracking attacks. If the adversary is not able to tell tags apart, he will not be able to track tags along the supply chain. We call this requirement *tag unlinkability* in accordance with [4, 15]. Notice that tag unlinkability is a stronger requirement than tag confidentiality. If an adversary $\mathcal{A}$ is able to jeopardize tag confidentiality, then he is automatically able to tell tags apart. Consequently, if CHECKER ensures tag unlinkability, then it ensures tag confidentiality as well.

2.) An adversary $\mathcal{A}$ must not be able to learn any information about the path of a tag T_i in the supply chain. Such a requirement ensures the privacy of the internal processes of the supply chain. Being unable to disclose any information about the path that tags took, the adversary cannot tell the origin of a tag T_i he is having access to, either the steps that T_i went through or the pallet of tags that T_i belongs to. In [4], this privacy requirement was captured by the notion of *step unlinkability*. More precisely, given two tags T_i and T_j, an adversary $\mathcal{A}$ must not be able to tell whether $\texttt{P}_i \cap \texttt{P}_j = \{\texttt{v}_0\}$ or not, where $\texttt{P}_i$ and $\texttt{P}_j$ denote the paths of tags T_i and T_j respectively. Observe that, all tags are issued by issuer I and thus, they all go through step $\texttt{v}_0$. For further details on step unlinkability, the interested reader may refer to Appendix A.1.

However, the definitions of tag unlinkability and step unlinkability as presented in [4] have two limitations:

1.) It is assumed that the manager M performing path verification cannot be corrupt. In this paper, path verification is performed by readers along the supply chain and these readers can behave arbitrarily, i.e., can be corrupt.

2.) It is assumed that adversary $\mathcal{A}$ has only a random access to tags in the supply chain. That is, $\mathcal{A}$ cannot choose tags he wants from $\mathcal{T}$. In this work, adversary $\mathcal{A}$ has more freedom in picking tags through the oracle $\mathcal{O}_{\texttt{Draw}}$. We recall that $\mathcal{A}$ can query the oracle $\mathcal{O}_{\texttt{Draw}}$ with a set of conditions $\texttt{c}_i$, and $\mathcal{O}_{\texttt{Draw}}$ has to return a tag T satisfying these conditions if there is any.

To address these limitations, we extend the privacy definitions of [4] by considering a more realistic adversary $\mathcal{A}$ who is allowed to corrupt readers and to select tags according to some conditions determined by him through the oracle $\mathcal{O}_{\texttt{Draw}}$.

One result of our modifications to privacy definitions is proving that if CHECKER ensures tag unlinkability, then it will as well ensure step unlinkability, see Appendix A.2 for a thorough analysis. Henceforth, we only focus on tag unlinkability.

Tag unlinkability

Read/write only tags cannot perform any computation. As a result, a tag T_i in CHECKER relies on readers in the supply chain to update its state, i.e., T_i's state does not change in between two protocol executions. Therefore, it is impossible to ensure tag unlinkability against an adversary who monitors all of T_i's interactions. Accordingly, there has to be at least *one unobserved* interaction between T_i and an *honest* reader outside the range of the adversary $\mathcal{A}$. This is in compliance with previous work dealing with read/write only tags, see Ateniese et al. [1], Dimitrou [11], Sadeghi et al. [19] and Blass et al. [4].

However, this assumption alone is not sufficient to ensure tag unlinkability against readers R_k along the supply chain. Notice that the genuineness verification of tags require readers R_k to have access to tags' identifiers or tags' pseudonyms. Although, adversary $\mathcal{A}$ does not observe all of T_i's interaction, he will be always able to link the interactions of tag T_i with corrupt readers.

Thus, we consider that adversary $\mathcal{A}$ is successful in mounting an attack against tag unlinkability if he is able to distinguish between two tags $\texttt{T}_0$ and $\texttt{T}_1$ which are not present at corrupt readers, and if $\texttt{T}_0$ and $\texttt{T}_1$ had at least one interaction with an honest reader R_i outside the range of $\mathcal{A}$.

We illustrate tag unlinkability by an experiment depicted in Algorithm 3 and Algorithm 4.

In the learning phase, $\mathcal{A}(r, s, \rho, \epsilon)$ can call the oracle $\mathcal{O}_{\texttt{Corrupt}}$ to corrupt up to r readers R_i. $\mathcal{A}$ is provided then with two challenge tags $\texttt{T}_0$ and $\texttt{T}_1$ that just entered the supply chain (tags at step $\texttt{v}_0$) from the oracle $\mathcal{O}_{\texttt{Draw}}$. Adversary $\mathcal{A}$ starts iterating the supply chain up to ρ times.

Before each iteration of the supply chain, $\mathcal{A}$ can read and write into tags $\texttt{T}_0, \texttt{T}_1$. He can also query the oracle $\mathcal{O}_{\texttt{Step}}$ to get the next steps of tags $\texttt{T}_0$ and $\texttt{T}_1$. Moreover, the oracle $\mathcal{O}_{\texttt{Draw}}$ supplies $\mathcal{A}$ with s tags $T_{(i,j)}$ fulfilling some condition $\texttt{c}_{(i,j)}$. $\mathcal{A}$ can read from and write into tags $T_{(i,j)}$. $\mathcal{A}$ is also supplied with the next step of tags $T_{(i,j)}$. $\mathcal{A}$ then iterates the supply chain and reads the state stored into tags $T_{(i,j)}$.

In the challenge phase, cf., Algorithm 4, $\mathcal{A}$ is provided with the next step of tags $\texttt{T}_0$ and $\texttt{T}_1$. He is also allowed to read and write into $\texttt{T}_0$ and $\texttt{T}_1$ one more time. Then, the supply chain is iterated first outside the range of $\mathcal{A}$. That is, tags $\texttt{T}_0$ and $\texttt{T}_1$ has an unobserved interaction with an honest reader outside the range of $\mathcal{A}$.

The oracle $\mathcal{O}_{\texttt{Flip}}$ supplies $\mathcal{A}$ with the tag $\texttt{T}_b$ which $\mathcal{A}$ can read. At the end of the challenge phase, $\mathcal{A}$ is required to output his guess of bit b.

$\mathcal{A}$ is said to be successful if **i.)** his guess of b is correct, **ii.)** the readers associated with steps $\texttt{v}^{k+1}_{\texttt{T}_0}$ and $\texttt{v}^{k'+1}_{\texttt{T}_1}$ are not corrupt, and **iii.)** the reader associated with the next step of tag $\texttt{T}_b$ at the end of the challenge phase is not corrupt by $\mathcal{A}$.

DEFINITION 2 (TAG UNLINKABILITY). CHECKER *provides tag unlinkability* $\Leftrightarrow$ *For adversary* $\mathcal{A}$, *inequality* $Pr(\mathcal{A}$ *is successful*$) \leq \frac{1}{2} + \epsilon$ *holds, where* ϵ *is negligible.*

In a real world scenario, the adversary $\mathcal{A}$ against the above experiment corresponds to a set of r partners $\{P_1, P_2, ..., P_r\}$ in the supply chain that collude in order to compromise the privacy of another partner P, through eavesdropping and tampering with tags

for $i := 1$ **to** r **do**
  $S_i \leftarrow \mathcal{O}_{\texttt{Corrupt}}(R_i)$;
end
$\texttt{T}_0 \leftarrow \mathcal{O}_{\texttt{Draw}}$(“tag at step ”$\texttt{v}_0$);
$\texttt{T}_1 \leftarrow \mathcal{O}_{\texttt{Draw}}$(“tag at step ”$\texttt{v}_0$);
for $i := 0$ **to** $\rho - 1$ **do**
  $\texttt{v}^{i+1}_{\texttt{T}_0} \leftarrow \mathcal{O}_{\texttt{Step}}(\texttt{T}_0)$;
  $s^i_{\texttt{T}_0}$:=READ($\texttt{T}_0$);
  WRITE($\texttt{T}_0, s'^i_{\texttt{T}_0}$);
  $\texttt{v}^{i+1}_{\texttt{T}_1} \leftarrow \mathcal{O}_{\texttt{Step}}(\texttt{T}_1)$;
  $s^i_{\texttt{T}_1}$:=READ($\texttt{T}_1$);
  WRITE($\texttt{T}_1, s'^i_{\texttt{T}_1}$);
  for $j = 1$ **to** s **do**
    $T_{(i,j)} \leftarrow \mathcal{O}_{\texttt{Draw}}(\texttt{c}_{(i,j)})$;
    $\texttt{v}_{T_{(i,j)}} \leftarrow \mathcal{O}_{\texttt{Step}}(T_{(i,j)})$;
    $s_{T_{(i,j)}}$:=READ($T_{(i,j)}$);
    WRITE($s_{T_{(i,j)}}, s'_{T_{(i,j)}}$);
  end
  ITERATESUPPLYCHAIN;
  for $j = 1$ **to** s **do**
    READ($T_{(i,j)}$);
  end
end

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

$\texttt{v}^{k+1}_{\texttt{T}_0} \leftarrow \mathcal{O}_{\texttt{Step}}(\texttt{T}_0)$;
$s^k_{\texttt{T}_0}$:=READ($\texttt{T}_0$);
WRITE($\texttt{T}_0, s'^k_{\texttt{T}_0}$);
$\texttt{v}^{k'+1}_{\texttt{T}_1} \leftarrow \mathcal{O}_{\texttt{Step}}(\texttt{T}_1)$;
$s^{k'}_{\texttt{T}_1}$:=READ($\texttt{T}_1$);
WRITE($\texttt{T}_1, s'^{k'}_{\texttt{T}_1}$);
ITERATESUPPLYCHAIN; *// Outside the range of $\mathcal{A}$*
$\texttt{T}_b \leftarrow \mathcal{O}_{\texttt{Flip}}\{\texttt{T}_0, \texttt{T}_1\}$;
s_{T_b} :=READ(T_b);
OUTPUT b;

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

present at P's site.

Note on tag unlinkability. The adversary $\mathcal{A}$ defined above is a narrow adversary as defined by Vaudenay [22]. That is, $\mathcal{A}$ does not have access to the output of the protocol in the challenge phase. In CHECKER's case, this corresponds to not accessing the result of the CHECK function. Note that if we allow $\mathcal{A}$ to have access to the output of the CHECK function, $\mathcal{A}$ can mount a trivial attack where he writes garbage, i.e., “dummy data” into a tag T_i. Tag T_i will not be accepted by any reader in the supply chain with high probability, and thus $\mathcal{A}$ can always distinguish T_i from legitimate tags.

4. PROTOCOL

Protocol overview

In CHECKER, a tag T stores a state s^j_T which consists of the *encryption* of T's identifier $\texttt{ID}$ and the *encryption* of a *path signature* that encodes the sequence of steps that T has visited.

To efficiently encode paths in the supply chain, we rely on a polynomial-based representation as introduced by Blass et al. [4]. That is, each path $\texttt{P}$ in the supply chain will match the evaluation of a unique polynomial $Q_{\texttt{P}}$ in a fixed value x_0, i.e., a path $\texttt{P}$ in the supply chain is mapped to $Q_{\texttt{P}}(x_0) \in \mathbb{F}_q$.

A tag T going through a valid path $\texttt{P}$ stores a randomly encrypted state $s^j_T = (\texttt{Enc}(\texttt{ID}), \texttt{Enc}(\sigma_{\texttt{P}}(\texttt{ID})))$, such that $\texttt{ID}$ is T's identifier, $\sigma_{\texttt{P}}(\texttt{ID}) = H(\texttt{ID})^{Q_{\texttt{P}}(x_0)}$, and H is a cryptographic hash function. The state s^j_T could be regarded as a message $\texttt{ID}$ and a signature on this message using the secret key $Q_{\texttt{P}}(x_0)$.

In CHECKER, the issuer I initializes a tag T by writing an initial encrypted state s^0_T. A reader R_k in CHECKER reads the encrypted state s^j_T stored into T and decrypts it using its secret key $\texttt{sk}_k$ to get the pair $(\texttt{ID}, \sigma_{\texttt{P}}(\texttt{ID}))$. R_k then uses its set of ν_k verification keys $\mathcal{K}^V_k = \{K^1_k, K^2_k, ..., K^{\nu_k}_k\}$ to verify whether T went through a valid path leading to R_k or not. After path verification, reader R_k uses an update function f_k to update the state stored into tag T accordingly. Finally, R_k encrypts the new state of tag T using the public key of T's next step.

Privacy and security overview

To protect *privacy* of tags in the supply chain against readers, tags store an IND-CCA secure encryption of their states. For ease of presentation, we use Cramer-Shoup's scheme (CS for short) [8] as the underlying encryption. As CHECKER takes place in subgroups of elliptic curves that support bilinear pairings, we note that any IND-CCA secure scheme that takes place in DDH-hard groups can be used to encrypt the tag state. Furthermore, readers in the supply chain do not share the same CS pair of keys, instead each reader R_k is equipped with a matching pair of CS public and secret keys $(\texttt{sk}_k, \texttt{pk}_k)$.

To ensure security, a tag T in CHECKER stores a signature of its $\texttt{ID}$ using the polynomial-based encoding of the path it took. Without having access to the polynomial-based encoding of valid paths, an adversary cannot forge a valid state; otherwise, we show that there is an adversary who is able to break the bilinear computational Diffie-Hellman (BCDH) assumption.

First, we introduce some of the definitions, notations and assumptions that will be used in the rest of the paper.

4.1 Preliminaries

CHECKER takes place in subgroups of elliptic curves that support bilinear pairings. Similar to related work on elliptic curves supporting bilinear pairings, we use multiplicative group notation [1, 2, 5]. If $\mathbb{G}$ is a subgroup of order q of some elliptic curve $\mathcal{E}$, then for all $g \in \mathbb{G}$ and $x \in \mathbb{Z}_q$, g^x denotes point multiplication of g by x.

4.1.1 Bilinear pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups, such that $\mathbb{G}_1$ and $\mathbb{G}_T$ have the same order q.

A pairing e: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear pairing if:

1. e is *bilinear*: $\forall\, x, y \in \mathbb{Z}_q$, $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, $e(g^x, h^y) = e(g, h)^{xy}$;
2. e is *computable*: there is an efficient algorithm to compute $e(g, h)$ for any $(g, h) \in \mathbb{G}_1 \times \mathbb{G}_2$;
3. e is *non-degenerate*: if g is a generator of $\mathbb{G}_1$ and h is a generator of $\mathbb{G}_2$, then $e(g, h)$ is a generator $\mathbb{G}_T$.

CHECKER's security and privacy rely on the Bilinear Computational Diffie-Hellman (BCDH) assumption and the Symmetric External Diffie-Hellman (SXDH) assumption [3, 20].

DEFINITION 3 (BCDH ASSUMPTION). *Let g be a generator of $\mathbb{G}_1$ and h be a generator of $\mathbb{G}_2$. We say that the Bilinear Computational Diffie-Hellman assumption holds if, given $g, g^x, g^y, g^z \in \mathbb{G}_1$ and $h, h^x, h^y \in \mathbb{G}_2$ for random $x, y, z \in \mathbb{F}_q$, the probability to compute $e(g, h)^{xyz}$ is negligible.*

DEFINITION 4 (SXDH ASSUMPTION). *The Symmetric External Diffie-Hellman assumption holds if $\mathbb{G}_1$ and $\mathbb{G}_2$ are two groups with the following properties:*

1. *There exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$;*
2. *the decisional Diffie-Hellman problem (DDH) is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Hence, CHECKER uses bilinear groups where DDH is hard, see Ateniese et al. [1, 2], Ballard et al. [3], Scott [20]. These groups can be chosen as specific subgroups of non supersingular elliptic curves such as Miyaji-Nakabayashi-Takano (MNT for short) curves [16]. Moreover, results by Galbraith et al. [13] indicate that these elliptic curves are the most efficient setting to implement pairing-based cryptography.

4.1.2 Polynomial-based path encoding

In this section, we briefly recall the polynomial-based path encoding as presented in [4]. In a nutshell, each step $\mathtt{v}_i,\ 0 \leq i \leq \eta$, in the supply chain is associated with a unique random number $a_i \in \mathbb{F}_q$, where q is a large prime ($|q| = 160$ bits).

Each path in the supply chain is mapped to a unique polynomial in $\mathbb{F}_q$. The polynomial corresponding to path $\mathtt{P} = \overrightarrow{\mathtt{v}_0\mathtt{v}_1 ... \mathtt{v}_l}$ is defined as:

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

To have a compact representation of paths, a path $\mathtt{P}$ is encoded as the evaluation of $Q_{\mathtt{P}}$ at x_0, where x_0 is a generator of $\mathbb{F}_q^*$. Consequently, providing an efficient encoding of paths that does not depend on the length of the paths.

We point out that when the coefficients a_i are chosen randomly in $\mathbb{F}_q$, then the above encoding has the following property: for any two different paths $\mathtt{P}$ and $\mathtt{P}'$, $Q_{\mathtt{P}}(x_0) \neq Q_{\mathtt{P}'}(x_0)$ with high probability, see [17] for more details.

In the remainder of this paper, we denote $\phi(\mathtt{P}) = Q_{\mathtt{P}}(x_0)$ the polynomial-based encoding of path $\mathtt{P}$.

For all paths $\mathtt{P}$ and for all steps $\mathtt{v}_k$ in the supply chain, the following holds:

$$\phi(\overrightarrow{\mathtt{P}\mathtt{v}_k}) = x_0 \cdot \phi(\mathtt{P}) + a_k$$

4.1.3 Path signature in CHECKER

Let T be a tag with the unique identifier $\mathtt{ID} \in \mathbb{G}_1$ that went through the path $\mathtt{P} = \overrightarrow{\mathtt{v}_0\mathtt{v}_1 ... \mathtt{v}_l}$. In CHECKER we define the *path signature* of tag T as:

$$\sigma_{\mathtt{P}}(\mathtt{ID}) = H(\mathtt{ID})^{\phi(\mathtt{P})}$$

where H is a cryptographic hash function $H : \mathbb{G}_1 \rightarrow \mathbb{G}_1$. For any $\mathtt{ID} \in \mathbb{G}_1$, such a hash function can be computed using the algorithms proposed by Icart [14] and Brier et al. [6]. In the security analysis, we view H as a random oracle.

Note that $\sigma_{\mathtt{P}}(\mathtt{ID})$ is a signature of $\mathtt{ID}$ using the secret key $\phi(\mathtt{P})$. More precisely, it is an aggregate signature using the secret coefficients a_i of readers R_i in path $\mathtt{P}$.

The identifier $\mathtt{ID}$ and the path signature $\sigma_{\mathtt{P}}(\mathtt{ID})$ are encrypted and stored into tag T. A reader R_k that is visited by tag T, decrypts T's state, verifies the validity of the state and updates the path signature $\sigma_{\mathtt{P}}(\mathtt{ID})$. Without loss of generality, we assume that T has gone through the path $\mathtt{P}$, and now it arrives at step $\mathtt{v}_{\mathtt{k}}$ in the supply chain. T stores the encrypted pair $(\mathtt{ID}, \sigma_{\mathtt{P}}(\mathtt{ID}))$ and $\mathtt{P}_k$ denotes the path $\overrightarrow{\mathtt{P}\mathtt{v}_k}$. To obtain $\sigma_{\mathtt{P}_k}(\mathtt{ID})$, reader R_k computes its state update function f_{R_k} defined as:

$$f_{R_k}(x, y) = x^{x_0} H(y)^{a_k} \quad (2)$$

Thus,

$$\begin{aligned} f_{R_k}(\sigma_{\mathtt{P}}(\mathtt{ID}), \mathtt{ID}) &= \sigma_{\mathtt{P}}(\mathtt{ID})^{x_0}\ H(\mathtt{ID})^{a_k} \\ &= H(\mathtt{ID})^{\phi(\mathtt{P})\cdot x_0}\ H(\mathtt{ID})^{a_k} \\ &= H(\mathtt{ID})^{x_0 \cdot \phi(\mathtt{P}) + a_k} \\ &= H(\mathtt{ID})^{\phi(\overrightarrow{\mathtt{P}\mathtt{v}_k})} = \sigma_{\mathtt{P}_k}(\mathtt{ID}) \end{aligned}$$

Therefore, we obtain the path signature of $\mathtt{P}_k = \overrightarrow{\mathtt{P}\mathtt{v}_k}$ from the path signature of $\mathtt{P}$.

4.1.4 Cramer-Shoup encryption

An elliptic curve Cramer-Shoup encryption consists of the following operations:

- *Setup:* The system outputs an elliptic curve $\mathcal{E}$ over a finite field $\mathbb{F}_p$. Let $\mathbb{G}_1$ be a subgroup of $\mathcal{E}$ of a large prime order q ($|q| = 160$ bits), where DDH is intractable. Let (g_1, g_2) be a pair of generators of the group $\mathbb{G}_1$.
- *Key generation:* The secret key is the random tuple $\mathtt{sk} = (x_1, x_2, y_1, y_2, z) \in \mathbb{F}_q^5$. The system computes then $(c, d, f) = (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^{z})$. Let G be a cryptographic hash function. The public key is $\mathtt{pk} = (g_1, g_2, c, d, f, G)$.
- *Encryption:* Given a message $m \in \mathbb{G}_1$, the encryption algorithm chooses $r \in \mathbb{F}_q$ at random. Then it computes $u_1 = g_1^r, u_2 = g_2^r, u = mf^r, \alpha = G(u_1, u_2, u), v = c^r d^{r\alpha}$. The encryption algorithm outputs the ciphertext $\mathtt{Enc}_{\mathtt{pk}}(m) = (u_1, u_2, u, v)$.
- *Decryption:* On input of a ciphertext $C = (u_1, u_2, u, v)$, the decryption algorithm first computes $\alpha = G(u_1, u_2, u)$, and tests if $v = u_1^{x_1 + y_1\alpha} u_2^{x_2 + y_2\alpha}$. If this condition does not hold, the decryption algorithm outputs $\perp$; otherwise, it outputs $\mathtt{Dec}_{\mathtt{sk}}(C) = \frac{u}{u_1^z}$.

4.2 Protocol description

CHECKER consists of an initial setup phase, the initialization of tags by the issuer, and finally the path verification and tag state update by the readers.

4.2.1 Setup

A trusted third party (TTP) outputs $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, h, H, G, e)$, where $\mathbb{G}_1$, $\mathbb{G}_T$ are subgroups of prime order q. g_1 and g_2 are random generators of $\mathbb{G}_1$. h is a generator of $\mathbb{G}_2$. $H : \mathbb{G}_1 \rightarrow \mathbb{G}_1$ is a secure hash function. $G : \mathbb{G}_1^3 \rightarrow \mathbb{F}_q$ is a secure hash function, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear pairing.

The TTP generates $\eta+1$ pairs of matching public and secret keys for the Cramer-Shoup encryption: $\mathtt{sk}_k = (x_{(1,k)}, x_{(2,k)}, y_{(1,k)}, y_{(2,k)}, z_k) \in \mathbb{F}_q^5$ and $\mathtt{pk}_k = (g_1, g_2, c_k, d_k, f_k, G)$, $0 \leq k \leq \eta$. The TTP generates as well $\eta + 1$ random coefficients $a_k \in \mathbb{F}_q$. Then, it selects a generator x_0 of $\mathbb{F}_q$.

Through a secure channel, the TTP sends to each reader $R_k, 1 \leq k \leq \eta$, the tuple $(x_0, a_k, \mathtt{sk}_k, \mathtt{pk}_k, H)$ and sends the tuple $(x_0, a_0, \mathtt{sk}_0, \mathtt{pk}_0, H)$ to the issuer I.

The TTP computes the verification keys for each reader R_k in the supply chain. Let $\mathtt{P}_k$ be a path leading to reader R_k. To obtain the verification key corresponding to path $\mathtt{P}_k$, the TTP computes the path encoding $\phi(\mathtt{P}_k)$. Then, the TTP outputs the corresponding verification key $K(\mathtt{P}_k) = h^{\phi(\mathtt{P}_k)} \in \mathbb{G}_2$.

Once the verification keys are computed, the TTP provides each reader R_k with its set $\mathcal{K}_V^k$ of verification keys.

We assume that the public keys $\mathtt{pk}_k, 0 \leq k \leq \eta$, are known to all parties in the system.

4.2.2 Tag initialization

For each new tag T in the supply chain, I chooses a random identifier $\mathtt{ID} \in \mathbb{G}_1$. The issuer computes the hash $H(\mathtt{ID})$, and using his secret coefficient a_0, he computes $H(\mathtt{ID})^{a_0}$. Provided with the public key of T's next step, the issuer computes a CS encryption of both ID and $\sigma_{\mathtt{v}_0}(\mathtt{ID}) = H(\mathtt{ID})^{a_0}$. Without loss of generality, we assume that T's next step is $\mathtt{v}_1$. The public key of step $\mathtt{v}_1$ is $\mathtt{pk}_1 = (g_1, g_2, c_1, d_1, f_1, G)$.

Issuer I draws two random number $r_{\mathtt{ID}}$ and r_σ in $\mathbb{F}_q$ and computes the following ciphertexts:

$$\begin{aligned}
c^0_{\mathtt{ID}} &= \mathtt{Enc}_{\mathtt{pk}_1}(\mathtt{ID}) = (u_{(1,\mathtt{ID})}, u_{(2,\mathtt{ID})}, u_{\mathtt{ID}}, v_{\mathtt{ID}}) \\
&= (g_1^{r_{\mathtt{ID}}}, g_2^{r_{\mathtt{ID}}}, \mathtt{ID}\, f_1^{r_{\mathtt{ID}}}, c_1^{r_{\mathtt{ID}}} d_1^{r_{\mathtt{ID}}\alpha_{\mathtt{ID}}}) \\
\alpha_{\mathtt{ID}} &= G(u_{(1,\mathtt{ID})}, u_{(2,\mathtt{ID})}, u_{\mathtt{ID}}) \\
c^0_\sigma &= \mathtt{Enc}_{\mathtt{pk}_1}(\sigma_{\mathtt{v}_0}(\mathtt{ID})) = (u_{(1,\sigma)}, u_{(2,\sigma)}, u_\sigma, v_\sigma) \\
&= (g_1^{r_\sigma}, g_2^{r_\sigma}, \sigma_{\mathtt{v}_0}(\mathtt{ID})\, f_1^{r_\sigma}, c_1^{r_\sigma} d_1^{r_\sigma \alpha_\sigma}) \\
\alpha_\sigma &= G(u_{(1,\sigma)}, u_{(2,\sigma)}, u_\sigma)
\end{aligned}$$

Finally, I writes state $s^0_T = (c^0_{\mathtt{ID}}, c^0_\sigma) \in \mathbb{G}_1^8$ into tag T. T then enters the supply chain.

4.2.3 Path verification by readers

Assume a tag T arrives at steps $\mathtt{v}_k$ in the supply chain. The reader R_k associated with step $\mathtt{v}_k$ reads the state $s^j_T = (c^j_{\mathtt{ID}}, c^j_\sigma)$ stored in tag T. Without loss of generality, we assume T went through path P. R_k using its secret key $\mathtt{sk}_k$ decrypts the CS ciphertexts $c^j_{\mathtt{ID}}$ and c^j_σ and gets respectively the pair $(\mathtt{ID}, \sigma_{\mathtt{P}}(\mathtt{ID}))$.

Let $\mathcal{K}_V^k$ denote the set of verification keys $\mathcal{K}_V^k = \{K_k^1, K_k^2, ..., K_k^{\nu_k}\} = \{h^{\phi(\mathtt{P}_k^1)}, h^{\phi(\mathtt{P}_k^2)}, ..., h^{\phi(\mathtt{P}_k^{\nu_k})}\}$ corresponding to the valid paths leading to step $\mathtt{v}_k$.

To verify whether the tag T went through a valid path or not, R_k computes the hash $H(\mathtt{ID})$ and checks whether there exists $i \in \{1, 2, ..., \nu_k\}$, such that:

$$\begin{aligned}
e(\sigma_{\mathtt{P}}(\mathtt{ID}), h) &= e(H(\mathtt{ID}), K_k^i) \\
&= e(H(\mathtt{ID}), h^{\phi(\mathtt{P}_k^i)})
\end{aligned}$$

If so, this implies that T went through a valid path leading to step $\mathtt{v}_k$. Otherwise, the reader concludes that tag T is illegitimate and rejects T.

4.2.4 Tag state update by readers

If the verification succeeds, the reader R_k in the supply chain is required to update the state of tag T. Using the update function f_{R_k}, the reader computes the new path signature $\sigma_{\overrightarrow{\mathtt{Pv_k}}}(\mathtt{ID})$.

$$\begin{aligned}
f_{R_k}(\sigma_{\mathtt{P}}(\mathtt{ID}), \mathtt{ID}) &= \sigma_{\mathtt{P}}(\mathtt{ID})^{x_0}\, H(\mathtt{ID})^{a_k} \\
&= H(\mathtt{ID})^{x_0 \phi(\mathtt{P}) + a_k} = H(\mathtt{ID})^{\phi(\overrightarrow{\mathtt{Pv_k}})} \\
&= \sigma_{\overrightarrow{\mathtt{Pv_k}}}(\mathtt{ID})
\end{aligned}$$

Without loss of generality, we assume that the tag's next step is $\mathtt{v}_{k+1}$. The reader R_k prepares tag T for reader R_{k+1} by encrypting the pair $(\mathtt{ID}, \sigma_{\overrightarrow{\mathtt{Pv_k}}}(\mathtt{ID}))$ using the public key $\mathtt{pk}_{k+1} = (g_1, g_2, c_{k+1}, d_{k+1}, f_{k+1}, G)$. Reader R_k obtains therefore, two ciphertexts $c^{j+1}_{\mathtt{ID}}$ and c^{j+1}_σ.

Finally, R_k writes the state $s^{j+1}_T = (c^{j+1}_{\mathtt{ID}}, c^{j+1}_\sigma)$ into T.

5. SECURITY AND PRIVACY ANALYSIS

In this section, we state the main theorems regarding CHECKER's security and privacy.

5.1 Security analysis

THEOREM 1. CHECKER *is secure under the BCDH assumption in the random oracle model.*

PROOF. Assume there is an adversary $\mathcal{A}$ who breaks the security of CHECKER with a non negligible advantage ϵ, we build an adversary $\mathcal{A}'$ that uses $\mathcal{A}$ as a subroutine to break the BCDH assumption with a non negligible advantage ϵ'.

Let $\mathcal{O}_{\mathrm{BCDH}}$ be an oracle that selects randomly $x, y, z \in \mathbb{F}_q$, and returns $g, g^x, g^y, g^z \in \mathbb{G}_1$, and $h, h^x, h^y \in \mathbb{G}_2$.

Proof overview. If $\mathcal{A}$ has a non negligible advantage in breaking the security of CHECKER, then $\mathcal{A}$ will be able to output a challenge tag $\mathtt{T}_c$ that stores a valid encrypted state $s_{\mathtt{T}_c}$, and:

i.) $\exists\, R_k$ such that $\mathrm{CHECK}(R_k, \mathtt{T}_c) = 1$, i.e., there is a path $\mathtt{P}_k^j$ that corresponds to $\mathtt{T}_c$'state;

ii.) $\exists\, \mathtt{v} \in \mathtt{P}_k^j$ such that step v is not corrupted by $\mathcal{A}$;

iii.) $\mathtt{T}_c$ did not go through step v.

To break BCDH, adversary $\mathcal{A}'$ simulates a CHECKER system for $\mathcal{A}$ where he provides a step $\mathtt{v}_i$ in the supply chain with the tuple $(x_0, g^x, \mathtt{sk}_i, \mathtt{pk}_i)$ instead of the tuple $(x_0, a_i, \mathtt{sk}_i, \mathtt{pk}_i)$.

Without loss of generality, we assume in the rest of the proof that $\mathtt{v}_i = \mathtt{v}_0$ and that $\mathcal{A}$ corrupts all readers R_k (but not issuer I) in the supply chain.

Now, $\mathcal{A}'$ must convince $\mathcal{A}$ that $\mathtt{v}_0$ is associated with secret coefficient $a_0 = x$ that corresponds to the pair (g^x, h^x) received from the oracle $\mathcal{O}_{\mathrm{BCDH}}$. Accordingly, $\mathcal{A}'$ has to be able to compute $H(\mathtt{ID})^x$ only by knowing (g^x, h^x). To tackle this issue, $\mathcal{A}'$ simulates a random oracle $\mathcal{H}$ that computes the hash function H.

When $\mathcal{H}$ is queried in the learning phase with identifier $\mathtt{ID}_j$, $\mathcal{A}'$ picks a random number r_j and computes $H(\mathtt{ID}_j) = g^{r_j}$.

Before the challenge phase, $\mathcal{A}$ queries the random oracle $\mathcal{H}$ with an identifier $\mathtt{ID}_c$, where $\mathtt{ID}_c$ is the identifier of the challenge tag T_c. Simulating $\mathcal{H}$, $\mathcal{A}'$ picks a random number r_c, computes $H(\mathtt{ID}_c) = g^{zr_c}$, and returns $H(\mathtt{ID}_c)$ to $\mathcal{A}$.

In the challenge phase, $\mathcal{A}$ supplies $\mathcal{A}'$ with the challenge tag $\mathtt{T}_c$.

As adversary $\mathcal{A}$ has a non negligible advantage in the security experiment, the challenge tag $\mathtt{T}_c$ stores an encrypted valid state that corresponds to the pair $(\mathtt{ID}_c, \sigma_c)$ such that $\sigma_c = H(\mathtt{ID}_c)^{\phi(\mathtt{P}_{\mathrm{valid}})}$, and $\mathtt{T}_c$ did not go through step $\mathtt{v}_0$.

Using σ_c, $\mathcal{A}'$ is able to identify the path $\mathtt{P}_{\mathrm{valid}}$ that corresponds to the state of tag $\mathtt{T}_c$. We assume that $\mathtt{P}_{\mathrm{valid}} = \overrightarrow{\mathtt{v}_0\mathtt{P}}$, and we denote l the length of path $\mathtt{P}_{\mathrm{valid}}$.

By definition, $\phi(\mathtt{P}_{\mathrm{valid}}) = a_0 x_0^l + \phi(\mathtt{P}) = x x_0^l + \phi(\mathtt{P})$, and given σ_c and the encoding $\phi(\mathtt{P})$ of the sub-path P, $\mathcal{A}'$ computes:

$$\begin{aligned}
\frac{\sigma_c}{H(\mathtt{ID}_c)^{\phi(\mathtt{P})}} &= \frac{H(\mathtt{ID}_c)^{\phi(\mathtt{P}_{\mathrm{valid}})}}{H(\mathtt{ID}_c)^{\phi(\mathtt{P})}} = H(\mathtt{ID}_c)^{x x_0^l} \\
H(\mathtt{ID}_c)^x &= \left(\frac{H(\mathtt{ID}_c)^{\phi(\mathtt{P}_{\mathrm{valid}})}}{H(\mathtt{ID}_c)^{\phi(\mathtt{P})}}\right)^{\frac{1}{x_0^l}}
\end{aligned}$$

$\mathcal{A}'$ thus have access to $H(\mathtt{ID}_c)^x = (g^{zr_c})^x = g^{xzr_c}$, and accordingly, he computes $(g^{xzr_c})^{\frac{1}{r_c}} = g^{xz}$.

Finally, $\mathcal{A}'$ computes $e(g^{xz}, h^y) = e(g, h)^{xyz}$, and this breaks the BCDH assumption which leads to a contradiction.

Simulation of the random oracle $\mathcal{H}$. To respond to the queries to the random oracle $\mathcal{H}$, $\mathcal{A}'$ keeps a table T_H of tuples $(\mathtt{ID}_j, r_j, \mathtt{coin}(\mathtt{ID}_j), h_j)$ as explained below.

On a query $H(\mathtt{ID}_i)$, $\mathcal{A}'$ replies as follows:

1.) If there is a tuple $(\mathtt{ID}_i, r_i, \mathtt{coin}(\mathtt{ID}_i), h_i)$ that corresponds to $\mathtt{ID}_i$, then $\mathcal{A}'$ returns $H(\mathtt{ID}_i) = h_i$.

2.) If $\mathtt{ID}_i$ has never been queried before, then $\mathcal{A}'$ picks a random number $r_i \in \mathbb{F}_q$. $\mathcal{A}'$ flips a random coin $\mathtt{coin}(\mathtt{ID}_i) \in \{0,1\}$ such that: $\mathtt{coin}(\mathtt{ID}_i) = 1$ with probability p, and is equals to 0 with probability $1-p$. The probability p will be determined later. If $\mathtt{coin}(\mathtt{ID}_i) = 0$, then $\mathcal{A}'$ answers with $H(\mathtt{ID}_i) = g^{r_i}$. Otherwise, $\mathcal{A}'$ answers with $h_i = H(\mathtt{ID}_i) = (g^z)^{r_i}$. Finally, $\mathcal{A}'$ stores the tuple $(\mathtt{ID}_i, r_i, \mathtt{coin}(\mathtt{ID}_i), h_i)$ in table T_H.

Construction. We detail below how $\mathcal{A}'$ breaks the BCDH assumption.

- First, $\mathcal{A}'$ queries $\mathcal{O}_{\text{BCDH}}$ to receive $g, g^x, g^y, g^z \in \mathbb{G}_1$ and $h, h^x, h^y \in \mathbb{G}_2$. Then, $\mathcal{A}'$ simulates a CHECKER system:

  1.) $\mathcal{A}'$ generates $\eta+1$ pairs of matching CS public and secret keys $(\mathtt{sk}_k, \mathtt{pk}_k)$. Then, he generates η random coefficients a_k.

  2.) $\mathcal{A}'$ provides each reader R_k in CHECKER with the tuple $(x_0, a_k, \mathtt{sk}_k, \mathtt{pk}_k)$.

  3.) $\mathcal{A}'$ provides the issuer I with the tuple $(x_0, g^x, \mathtt{sk}_0, \mathtt{pk}_0)$, as if $a_0 = x$.

  4.) $\mathcal{A}'$ computes the verification keys for each reader R_k in the supply chain. Without loss of generality, a valid path $\mathtt{P}_{\text{valid}}$ in the supply chain could be represented as $\mathtt{P}_{\text{valid}} = \overrightarrow{\mathtt{v}_0\mathtt{P}'_{\text{valid}}}$. Thus, the corresponding verification key $K(\mathtt{P}_{\text{valid}})$ is computed as: $K(\mathtt{P}_{\text{valid}}) = (h^x)^{x_0^l} h^{\phi(\mathtt{P}'_{\text{valid}})} = h^{\phi(\mathtt{P}_{\text{valid}})}$, where l is the length of path $\mathtt{P}_{\text{valid}}$.

  Once the verification keys are computed for all the readers R_k, $\mathcal{A}$ provides each reader R_k with his set $\mathcal{K}_V^k$ of verification keys.

$\mathcal{A}'$ then calls the adversary $\mathcal{A}$.

- $\mathcal{A}'$ simulates the issuer I and creates n tags T_j of CHECKER.

  $\mathcal{A}'$ selects randomly $\mathtt{ID}_j \in \mathbb{G}_1$. He simulates the oracle $\mathcal{H}$. $\mathcal{A}'$ gets the tuple $(\mathtt{ID}_j, r_j, \mathtt{coin}(\mathtt{ID}_j), h_j)$.

  If $\mathtt{coin}(\mathtt{ID}_j) = 1$, i.e., $h_j = H(\mathtt{ID}_j) = g^{zr_j}$, then $\mathcal{A}'$ cannot compute $H(\mathtt{ID}_j)^x = g^{xzr_j}$ as he does not know both x and z. Consequently, $\mathcal{A}'$ stops the security experiment.

  Otherwise, using r_j $\mathcal{A}'$ computes $H(\mathtt{ID}_j)^x = (g^x)^{r_j}$.

  Finally, $\mathcal{A}'$ encrypts both $\mathtt{ID}_j$ and $\sigma_{\mathtt{v}_0}(\mathtt{ID}_j)$ using the public key of the tag T_j's next step. $\mathcal{A}'$ stores the resulting ciphertexts $(c^0_{(\mathtt{ID},j)}, c^0_{(\sigma,j)})$ into tag T_j.

- $\mathcal{A}'$ simulates the oracle $\mathcal{O}_{\mathtt{Corrupt}}$ for $\mathcal{A}$. For ease of understanding, we assume that $\mathcal{A}$ corrupts all readers R_k in the supply chain.

- $\mathcal{A}'$ simulates readers R_k along the supply chain. Let T_j be a tag which went through path $\mathtt{P}$ and arrives at step $\mathtt{v}_k$.

  $\mathcal{A}'$ decrypts the tag T_j's state using CS secret key $\mathtt{sk}_k$ of reader R_k and gets the pair $(\mathtt{ID}_j, \sigma_{\mathtt{P}}(\mathtt{ID}_j))$. He verifies the path of tag T_j using $\mathcal{K}_k^V$. Then, $\mathcal{A}'$ updates the path of tag T_j using the secret coefficient a_k.

  Then using the public key of T_j's next step, $\mathcal{A}'$ encrypts T_j's identifier and T_j's path signature.

- In the challenge phase, $\mathcal{A}$ outputs a tag $\mathtt{T}_c$.

- $\mathcal{A}'$ simulates all the readers in the supply chain and verifies whether the encrypted state stored into tag $\mathtt{T}_c$ matches a valid path in the supply chain. That is, $\mathcal{A}'$ verifies whether there exists a reader R_k in the supply chain that outputs CHECK$(R_k, T_c) = 1$ or not.

  Adversary $\mathcal{A}$ has a non negligible advantage in the security experiment, consequently, **i.)** $\exists\, R_k$ such that CHECK$(R_k, \mathtt{T}_c) = 1$, and **ii.)** $\mathtt{T}_c$ did not go through step $\mathtt{v}_0$.

We assume without loss of generality that T_c's state corresponds to the pair $(\mathtt{ID}_c, \sigma_c)$, and that $\mathtt{T}_c$'s path signature σ_c corresponds to path $\mathtt{P}_{\text{valid}} = \overrightarrow{\mathtt{v}_0\mathtt{P}}$.

- $\mathcal{A}'$ first checks whether $\mathtt{coin}(\mathtt{ID}_c) = 1$ or not.

  If $\mathtt{coin}(\mathtt{ID}_c) = 0$, then $\mathcal{A}'$ stops the experiment. Notice that if $h_c = H(\mathtt{ID}_c) = g^{r_c}$, $\mathcal{A}'$ will not be able to break the BCDH assumption.

  If $\mathtt{coin}(\mathtt{ID}_c) = 1$, i.e., $h_c = H(\mathtt{ID}_c) = g^{zr_c}$, then $\mathcal{A}'$ continues the experiment, and computes $e(g,h)^{xyz}$.

Let l denote the length of path $\mathtt{P}_{\text{valid}}$. Accordingly,

$$\phi(\mathtt{P}_{\text{valid}}) = a_0 x_0^l + \phi(\mathtt{P}) = x x_0^l + \phi(\mathtt{P})$$

and,

$$\begin{aligned} H(\mathtt{ID}_c)^{xx_0^l} &= \frac{\sigma_c}{H(\mathtt{ID}_c)^{\phi(\mathtt{P})}} = \frac{H(\mathtt{ID}_c)^{\phi(\mathtt{P}_{\text{valid}})}}{H(\mathtt{ID}_c)^{\phi(\mathtt{P})}} \\ H(\mathtt{ID}_c)^x &= \left(\frac{H(\mathtt{ID}_c)^{\phi(\mathtt{P}_{\text{valid}})}}{H(\mathtt{ID}_c)^{\phi(\mathtt{P})}}\right)^{\frac{1}{x_0^l}} \\ e(H(\mathtt{ID}_c)^x, h^y) &= e((g^{zr_c})^x, h^y) = e(g,h)^{xyzr_c} \end{aligned}$$

Provided with the random number r_c, $\mathcal{A}'$ finally computes

$$e(g,h)^{xyz} = (e(g,h)^{xyzr_c})^{\frac{1}{r_c}}$$

Here, we compute the advantage of $\mathcal{A}'$.

Notice that $\mathcal{A}'$ succeeds in breaking the BCDH assumption if he does not stop the security experiment.

1.) $\mathcal{A}'$ halts the experiment, if during the initialization phase of the n tags T_j of the CHECKER system, the simulated random oracle $\mathcal{H}$ returns $(\mathtt{ID}_j, r_j, \mathtt{coin}(\mathtt{ID}_j), h_j)$ such that $\mathtt{coin}(\mathtt{ID}_j) = 1$. This event occurs with probability p. Hence, the probability that $\mathcal{A}'$ does not stop the experiment during the learning phase is: $(1-p)^n$.

2.) $\mathcal{A}'$ stops the experiment during the challenge phase, if $\mathtt{coin}(\mathtt{ID}_c) = 0$. As a result, $\mathcal{A}'$ does not stop the experiment in the challenge phase with probability p.

Let E denote the event: $\mathcal{A}'$ does not abort the security experiment.

Let E_1 denote the event: $\mathcal{A}'$ does not abort security experiment in the learning phase, $Pr(E_1) = (1-p)^n$.

Let E_2 denote the event: $\mathcal{A}'$ does not abort security experiment in the challenge phase, $Pr(E_2) = p$. Hence,

$$\begin{aligned} \pi &= Pr(E) = Pr(E_1)Pr(E_2) \\ &= p(1-p)^n \end{aligned}$$

Now, if $\mathcal{A}$ has a non negligible advantage ϵ in breaking the security of CHECKER, then $\mathcal{A}'$ can break the BCDH assumption with advantage $\epsilon' = \pi\epsilon$, leading to a contradiction.

Remark that π is maximal when $p = \frac{1}{n}$ and $\pi_{\mathtt{max}} = \frac{(1-\frac{1}{n})^n}{n} \simeq \frac{1}{en}$. Consequently, the advantage ϵ' in breaking BCDH is in this case $\epsilon' = \frac{\epsilon}{en}$. $\square$

5.2 Privacy analysis

Tag unlinkability

THEOREM 2. CHECKER *provides tag unlinkability under the SXDH assumption.*

PROOF. To prove tag unlinkability, we use the IND-CCA property of Cramer-Shoup encryption ensured under the SXDH assumption.

Before presenting the proof, we introduce the definition of IND-CCA.

Let $\mathcal{O}_{\text{decryption}}$ be the oracle that, on input of a ciphertext c encrypted with public key $\texttt{pk}$, outputs the underlying plaintext m.

Let $\mathcal{O}_{\text{encryption}}$ be the oracle that, provided with two messages m_0, m_1 and public key $\texttt{pk}$, randomly chooses $b \in \{0,1\}$, encrypts m_b using public key $\texttt{pk}$, and returns the challenge ciphertext c_b.

Let $\mathcal{A}$ (r, s, ϵ) be an adversary that is allowed to make r calls to the oracle $\mathcal{O}_{\text{decryption}}$ with arbitrary ciphertexts c_i. Then, $\mathcal{A}$ selects two messages m_0 and m_1 which he provides to the oracle $\mathcal{O}_{\text{encryption}}$. $\mathcal{O}_{\text{encryption}}$ returns the challenge ciphertext c_b. After receiving c_b, $\mathcal{A}$ can still query the decryption oracle $\mathcal{O}_{\text{decryption}}$ with s ciphertexts c'_i, with the only restrictions that $c_b \neq c'_i$. Finally, $\mathcal{A}$ is required to output his guess of b. An encryption is IND-CCA secure, if $\mathcal{A}$ (r, s, ϵ) has a negligible advantage ϵ in outputting a correct guess of b.

Assume there is an adversary $\mathcal{A}$ who breaks the security of CHECKER with a non negligible advantage ϵ, we show that there is an adversary $\mathcal{A}'$ that uses $\mathcal{A}$ as a subroutine and breaks the IND-CCA property of Cramer-Shoup encryption with a non-negligible advantage ϵ'.

Proof overview. The idea of the proof is to build a CHECKER system such that there is a step $\texttt{v}_i$ in the supply chain that is associated with the public key $\texttt{pk}$, where $\texttt{pk}$ is the challenge public key from the IND-CCA security experiment.

In the learning phase, $\mathcal{A}'$ is required to simulate reader R_i. This implies that $\mathcal{A}'$ has to be able decrypt the state of tags arriving at step $\texttt{v}_i$. Hence the need to a decryption oracle and therewith to an IND-CCA secure encryption. Now, whenever a tag T arrives at step $\texttt{v}_i$, $\mathcal{A}'$ first calls the decryption oracle for the Cramer-Shoup encryption $\mathcal{O}_{\text{decryption}}$ that returns the underlying plaintexts, i.e., $(\texttt{ID}, \sigma_{\texttt{P}}(\texttt{ID}))$. Then, $\mathcal{A}'$ verifies the validity of the pair and updates the state of T accordingly.

In the challenge phase, $\mathcal{A}$ returns the challenge tags $\texttt{T}_0$ and $\texttt{T}_1$ to $\mathcal{A}'$. $\mathcal{A}'$ decrypts the state of tags $\texttt{T}_0$ and $\texttt{T}_1$ and gets their identifiers $\texttt{ID}_0$ and $\texttt{ID}_1$ respectively. Then, $\mathcal{A}'$ queries the encryption oracle $\mathcal{O}_{\text{encryption}}$ with message $\texttt{ID}_0$ and $\texttt{ID}_1$. $\mathcal{O}_{\text{encryption}}$ returns the challenge ciphertext $c_b = \texttt{Enc}_{\texttt{pk}}(\texttt{ID}_b), b \in \{0,1\}$. $\mathcal{A}'$ iterates the supply chain outside the range of $\mathcal{A}$, and simulates $\mathcal{O}_{\texttt{Flip}}$ by returning $\texttt{T}_b$ which stores the ciphertext c_b along with an encryption of $\texttt{T}_b$'s path signature. As $\mathcal{A}'$ makes a guess for the value of b to update T_b's path signature, this latter will be correct with probability $\frac{1}{2}$.

If $\mathcal{A}$ has a non-negligible advantage ϵ in breaking the tag unlinkability experiment, then he outputs a correct guess for the value of b. If $b = 0$, then this implies that $\texttt{T}_b$ stores an encryption of $\texttt{ID}_0$ and thus $c_b = \texttt{Enc}_{\texttt{pk}}(\texttt{ID}_0)$; otherwise, $c_b = \texttt{Enc}_{\texttt{pk}}(\texttt{ID}_1)$.

Construction. To break the IND-CCA property of Cramer and Shoup encryption, $\mathcal{A}'$ proceeds as follows:

- $\mathcal{A}'$ creates a supply chain for the CHECKER protocol.
- $\mathcal{A}'$ calls the adversary $\mathcal{A}$. $\mathcal{A}$ queries the oracle $\mathcal{O}_{\texttt{Corrupt}}$ with the identity of r readers R_k. $\mathcal{A}'$ simulates the oracle $\mathcal{O}_{\texttt{Corrupt}}$ and assigns to each reader R_k a tuple $(x_0, a_k, \texttt{sk}_k, \texttt{pk}_k)$ that he returns to $\mathcal{A}$.
- Now, $\mathcal{A}'$ selects a reader R_i from the set of uncorrupt readers and assigns to R_i the tuple $(x_0, a_i, \texttt{pk}_i = \texttt{pk})$. Without loss of generality, we assume that step $\texttt{v}_i$ in the supply chain is associated with reader R_i.
- Simulating $\mathcal{O}_{\texttt{Draw}}$, $\mathcal{A}'$ supplies $\mathcal{A}$ with two challenge tags $\texttt{T}_0$ and $\texttt{T}_1$ that have just been issued by issuer I, i.e., just entered the supply chain.
- $\mathcal{A}$ iterates the supply chain ρ times. Before each iteration j of the supply chain:

  1.) $\mathcal{A}$ reads and writes into tags $\texttt{T}_0$, $\texttt{T}_1$.

  2.) Simulating $\mathcal{O}_{\texttt{Step}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with the next step of tags $\texttt{T}_0$, $\texttt{T}_1$.

  3.) $\mathcal{A}'$ simulates $\mathcal{O}_{\texttt{Draw}}$ and supplies $\mathcal{A}$ with s tags $T_{(i,j)}$. $\mathcal{A}$ is also provided with $T_{(i,j)}$'s next step. $\mathcal{A}$ then iterates the supply chain and reads the states stored into tags $T_{(i,j)}$.
- If a tag T in the learning phase arrives at step $\texttt{v}_i$, then $\mathcal{A}'$ simulates reader R_i as follows:

  1.) $\mathcal{A}'$ reads the state stored into T and gets two CS ciphertexts $c_{\texttt{ID}}$ and c_σ.

  2.) $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\text{decryption}}$ with the ciphertexts $c_{\texttt{ID}}$ and c_σ. The oracle $\mathcal{O}_{\text{decryption}}$ returns the corresponding plaintexts $\texttt{ID}$ and σ.

  3.) $\mathcal{A}'$ checks then if the pair $(\texttt{ID}, \sigma)$ corresponds to a valid path leading to step $\texttt{v}_i$.

  4.) Finally, $\mathcal{A}'$ updates the path signature of T accordingly and encrypts both the identifier $\texttt{ID}$ and the path signature using the public key of T's next step.
- In the challenge phase, tags $\texttt{T}_0$ and $\texttt{T}_1$ are submitted to $\mathcal{A}'$. $\mathcal{A}'$ decrypts the states stored into $\texttt{T}_0$ and $\texttt{T}_1$, and gets $\texttt{ID}_0$ and $\texttt{ID}_1$ respectively.
- $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\text{encryption}}$ with messages $\texttt{ID}_0$ and $\texttt{ID}_1$. $\mathcal{O}_{\text{encryption}}$ returns $c_{(\texttt{ID},b)} = \texttt{Enc}_{\texttt{pk}}(\texttt{ID}_b)$.
- $\mathcal{A}'$ prepares the tag $\texttt{T}_b$ for the adversary $\mathcal{A}$:

  1.) $\mathcal{A}'$ updates the path of tags $\texttt{T}_0$ and $\texttt{T}_1$ and encrypts the path signature using the public key $\texttt{pk}$. He obtains two ciphertexts: $c_{(\sigma,0)}$ and $c_{(\sigma,1)}$.

  2.) $\mathcal{A}'$ randomly selects $b' \in \{0,1\}$ and stores the state $s_{T_b} = (c_{(\texttt{ID},b)}, c_{(\sigma,b')})$ in $\texttt{T}_b$. Therefore, $\texttt{T}_b$'s next step is step $\texttt{v}_i$ associated with public key $\texttt{pk}$.
- Simulating $\mathcal{O}_{\texttt{Flip}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with the challenge tag $\texttt{T}_b$. $\mathcal{A}$ is allowed to read from tag $\texttt{T}_b$.

Notice that if $b = b'$, then the state $s_{\texttt{T}_b} = (c_{(\texttt{ID},b)}, c_{(\sigma,b')})$ computed by $\mathcal{A}'$ when simulating CHECKER corresponds to a well formed pair $(\texttt{ID}_b, \sigma_{\texttt{P}}(\texttt{ID}_b))$, and consequently, the simulation of CHECKER by $\mathcal{A}'$ does not differ from an actual CHECKER system. $\mathcal{A}$ can accordingly output his guess for the tag corresponding to the challenge tag $\texttt{T}_b$ with a non-negligible advantage ϵ.

If $\mathcal{A}$ outputs $b = 0$, this means that $\texttt{T}_b$ stores an encryption of $\texttt{ID}_0$, and $\mathcal{A}'$ outputs 0. If $\mathcal{A}$ outputs $b = 1$, this means that $\texttt{T}_b$ stores an encryption of $\texttt{ID}_1$, and $\mathcal{A}'$ outputs 1.

If $b \neq b'$, then the probability that $\mathcal{A}'$ breaks the IND-CCA property of CS is at worst a random guess, i.e., $\frac{1}{2}$.

Now, we quantify the advantage of $\mathcal{A}'(q, 0, \epsilon')$, $(q \leq 2(s\rho + 2\rho + 2)$, in breaking the IND-CCA property of CS.

– Let E_1 be the event that $\mathcal{A}'$ breaks the IND-CCA property of CS.

– Let E_2 be the event that $b = b'$.

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

$$\begin{aligned} Pr(E_1) &= Pr(E_1|E_2) \cdot Pr(E_2) + Pr(E_1|\overline{E_2}) \cdot Pr(\overline{E_2}) \\ &= \frac{1}{2}Pr(E_1|E_2) + \frac{1}{2}Pr(E_1|\overline{E_2}) \\ &= \frac{1}{2}\left(\frac{1}{2} + \epsilon\right) + \frac{1}{2}Pr(E_1|\overline{E_2}) \\ &\geq \frac{1}{2}\left(\frac{1}{2} + \epsilon + \frac{1}{2}\right) = \frac{1}{2} + \frac{\epsilon}{2} \end{aligned}$$

Thus, the advantage of $\mathcal{A}'$ to break the IND-CCA property of CS is at least $\epsilon' = \frac{\epsilon}{2}$. Therefore, if $\mathcal{A}$ has a non-negligible advantage ϵ to break CHECKER, then $\mathcal{A}'(q, 0, \epsilon')$ will have a non-negligible advantage ϵ' to break the IND-CCA property of Cramer and Shoup encryption, which leads to a contradiction. □

6. RELATED WORK

Ouafi and Vaudenay [18] propose a solution that allows product tracking. However, this solution assumes that tags can perform hash functions. We argue that a wide implementation of such tracking systems requires using the cheapest kind of tags which correspond to read/write only tags.

Also Burbridge and Soppera [7] suggest the use of proxy re-signature to allow path segment verification while using read/write only tags. The tag stores a signature of the last trusted party it has visited. To prevent product injection in the supply chain, partners in the supply chain do not have secret keys to sign tags' identifiers, but rather secret proxy keys that only allow partners to transform a valid signature of one partner to their own signature. The paper however does not address the problem of implementing practical proxy re-signatures without trusted third party.

Although these two previous schemes ensure secure product tracking in the supply chain, they fail at providing a privacy preserving solution. We emphasize that any solution dealing with product/tag tracking should preserve tag privacy in order to protect the privacy of the internal processes of partners in the supply chain.

Blass et al. [4] propose a tracking system using read/write only tags that preserve tags' privacy in the supply chain. They use polynomial based path encoding to represent efficiently the paths in the supply chain, and they rely on the use of homomorphic and probabilistic encryption to preserve tag privacy against readers in the supply chain and external adversaries. However, contrary to the work at hand, [4] relies on the assumption that the path verification is carried out by a trusted party called manager M. Thus, it does not permit the readers in the supply chain to perform the path verification which is the main application scenario for CHECKER.

7. EVALUATION

CHECKER targets read/write only tags that do no feature any computational capabilities. A tag in CHECKER is required to store a pair of IND-CCA encryptions of its identifier ID and its path signature $\sigma_{\mathtt{P}_{\text{valid}}}(\mathtt{ID}) = H(\mathtt{ID})^{\phi(\mathtt{P}_{\text{valid}})}$. In this paper, we use Cramer-Shoup's scheme as the underlying encryption. This results in an overall tag storage of $2 \cdot 4 \cdot 160 = 1280$ bits. We emphasize that any IND-CCA secure encryption in DDH-hard subgroups of elliptic curve is sufficient to implement CHECKER. One possible choice of encryption scheme is CS-lite [8]: a light variant of CS encryption which is IND-CCA secure and costs 480 bits per encryption instead of 640 bits. Also, there is a variant of Elgamal proposed by Fujisaki and Okamoto [12] which is IND-CCA secure in the random oracle model, and whose storage requirements are comparable to Elgamal's.

We believe that CHECKER can be implemented in current ISO 18000-3 HF tags, such as UPM RFID MiniTrack tags [21] that feature 1 kbit of memory.

Moreover, a reader R_k in the supply chain is required to decrypt the state stored into tags using its secret key $\mathtt{sk}_k$, then to verify the validity of the paths that tags went through, and to update and encrypt the state of tags. This amounts to performing: **1.)** two decryptions in $\mathbb{G}_1$ where $|\mathbb{G}_1| = 160$ bits, **2.)** ν_k bilinear pairings' computation in $\mathbb{G}_T$, where ν_k is the number of verification keys of reader R_k and $|\mathbb{G}_T| = 1024$ bits, **3.)** two exponentiations in $\mathbb{G}_1$ to update the path signature, and finally **4.)** two encryptions in $\mathbb{G}_1$. The costly operation at reader R_k is the verification of the path signature which is linear in the number of valid paths leading to reader R_k. We can further decrease the computation load at the readers by allowing tags to store the verification key that corresponds to the path that they took in the supply chain.

We recall that a verification key of path $\mathtt{P}_{\text{valid}}$ is $h^{\phi(\mathtt{P}_{\text{valid}})} \in \mathbb{G}_2$. Now, instead of storing an encrypted pair $(\mathtt{ID}, H(\mathtt{ID})^{\phi(\mathtt{P}_{\text{valid}})})$, a tag T stores the encrypted tuple $(\mathtt{ID}, H(\mathtt{ID})^{\phi(\mathtt{P}_{\text{valid}})}, h^{\phi(\mathtt{P}_{\text{valid}})})$. When T arrives at step $\mathtt{v}_k$, the reader R_k decrypts T's state and gets a tuple (α, β, γ). First, R_k checks whether γ is in his set of verification keys $\mathcal{K}_V^k$ or not. If so, R_k proceeds in verifying the path signature of tag T. Consequently, the cost of the verification of the path signature at the readers is constant. On the one hand however, readers are required to perform an additional table lookup, one decryption, two exponentiations and one encryption in $\mathbb{G}_2$. On the other hand, tags have to store three encryptions of size 640 bits in the case of Cramer-Shoup, and of size 480 in the case of CS-lite.

8. CONCLUSION

In this paper, we presented CHECKER for a secure and privacy preserving product genuineness verification in supply chains. CHECKER relies solely on read/write RFID tags that do not feature any computational capabilities. CHECKER allows on-site checking by providing the readers with the means to verify the validity of the paths that products took. CHECKER's main idea is to sign the tag's identifier using the encoding of the path that the tag took. Then, each reader in CHECKER is supplied with a set of public keys that correspond to the set of valid paths in the supply chain. This grants readers the ability to verify the genuineness of products, while preventing them from injecting fake products. CHECKER's security and privacy rely on standard assumptions: the BCDH and the DDH assumptions. Finally, CHECKER does not involve a trusted party and therefore, it is well suited for the distributed and heterogeneous setting of supply chains.

Bibliography

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

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

[3] L. Ballard, M. Green, B. de Medeiros, and F. Monrose. Correlation-Resistant Storage via Keyword-Searchable En-

cryption. Cryptology ePrint Archive, Report 2005/417, 2005. http://eprint.iacr.org/.

[4] E-O Blass, K Elkhiyaoui, and R Molva. Tracker: security and privacy for RFID-based supply chains. In *NDSS'11, 18th Annual Network and Distributed System Security Symposium, 6-9 February 2011, San Diego, California, USA, ISBN 1-891562-32-0*, 02 2011.

[5] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17:297–319, 2004. ISSN 0933-2790.

[6] E. Brier, J.S. Coron, T. Icart, D. Madore, H. Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 237–254. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-14622-0.

[7] T. Burbridge and A. Soppera. Supply chain control using a RFID proxy re-signature scheme. In *RFID, 2010 IEEE International Conference on*, pages 29 –36, april 2010.

[8] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO '98*, pages 13–25. Springer-Verlag, 1998.

[9] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-14576-6.

[10] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-17372-1.

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

[12] E. Fujisaki and T. Okamoto. How to Enhance the Security of Public-Key Encryption at Minimum Cost. In *Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography*, PKC '99, pages 53–68, London, UK, 1999. Springer-Verlag. ISBN 3-540-65644-8.

[13] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156:3113–3121, September 2008. ISSN 0166-218X.

[14] T. Icart. How to Hash into Elliptic Curves. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-03355-1.

[15] 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.

[16] A. Miyaji, M. Nakabayashi, and S. Takano. New Explicit Conditions of Elliptic Curve Traces for FR-Reduction. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 2001.

[17] 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.

[18] 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.

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

[20] M. Scott. Authenticated ID-based Key Exchange and remote log-in with simple token and PIN number. Cryptology ePrint Archive, Report 2002/164, 2002. http://eprint.iacr.org/.

[21] UPM RFID Technology. UPM Raflatac MiniTrak datasheet, 2011. http://www.upmrfid.com/rfid/images/MiniTrack_SLI_datasheet.pdf/$FILE/MiniTrack_SLI_datasheet.pdf.

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

APPENDIX

A. STEP UNLINKABILITY

A.1 Definition

As explained in Section 3.2, step unlinkability captures the ability of an adversary $\mathcal{A}$ of telling if the paths of two tags $\mathtt{T}_0$ and $\mathtt{T}_1$ have a step in common besides the step $\mathtt{v}_0$. Notice that step unlinkability as defined hereafter makes sense only when the system comprises at least two tags.

We use an experiment based definition as in Section 3.2. In addition to the oracles presented earlier, $\mathcal{A}$ has access to the oracle $\mathcal{O}_{\mathtt{Path}}$. When $\mathcal{O}_{\mathtt{Path}}$ is queried with a path $\mathtt{P}$, it flips a fair coin $b \in \{0, 1\}$. If $b = 1$, then $\mathcal{O}_{\mathtt{Path}}$ selects randomly a tag T which is going through a step $\mathtt{v} \in \mathtt{P} \setminus \{\mathtt{v}_0\}$ in the next supply chain iteration. Otherwise, $\mathcal{O}_{\mathtt{Path}}$ selects randomly a tag T which is going through a step $\mathtt{v} \notin \mathtt{P}$. Finally, $\mathcal{O}_{\mathtt{Path}}$ returns the tag T to $\mathcal{A}$.

An adversary $\mathcal{A}(r, s, t, \rho, \epsilon)$ against step unlinkability has access to CHECKER in two phases. In the learning phase as illustrated in Algorithm 5, $\mathcal{A}$ calls the oracle $\mathcal{O}_{\mathtt{corrupt}}$ that furnishes $\mathcal{A}$ with the secret information of r readers R_i of his choice. Now, $\mathcal{A}$ controls steps $\mathtt{v}_i$ associated with readers R_i. $\mathcal{A}$ then queries the oracle $\mathcal{O}_{\mathtt{Draw}}$ which supplies $\mathcal{A}$ with a tag $\mathtt{T}_0$ entering the supply chain.

$\mathcal{A}$ is allowed to iterate the supply chain up to ρ times. Before each iteration i of the supply chain, $\mathcal{A}$ can read and re-write the internal state of tag $\mathtt{T}_0$. He also queries the oracle $\mathcal{O}_{\mathtt{Step}}$ that returns the next step $\mathtt{v}_{\mathtt{T}_0}^{i+1}$ of tag $\mathtt{T}_0$ in the supply chain. $\mathcal{A}$ then calls the oracle $\mathcal{O}_{\mathtt{Draw}}$ which gives $\mathcal{A}$ s tags $T_{(i,j)}$, that are going through $\mathtt{v}_{\mathtt{T}_0}^{i+1}$ in the next supply chain iteration. Also, $\mathcal{A}$ can query the oracle $\mathcal{O}_{\mathtt{Draw}}$ again to provide him with t other tags $T'_{(i,j)}$, that fulfill some condition $\mathtt{c}_{(i,j)}$ specified by $\mathcal{A}$. Now, $\mathcal{A}$ has a full access to these tags, i.e., $\mathcal{A}$ can read from and write into them, he can as well have access to the next step of tags $T'_{(i,j)}$. Finally, $\mathcal{A}$ iterates the supply chain by calling ITERATESUPPLYCHAIN and reads the state stored into the tags $T_{(i,j)}$ and $T'_{(i,j)}$.

Let $\mathtt{P}_{\mathtt{T}_0}$ denote the path that tag $\mathtt{T}_0$ went through.

In the challenge phase, cf., Algorithm 6, $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathtt{Path}}$ with path $\mathtt{P}_{\mathtt{T}_0}$. $\mathcal{O}_{\mathtt{Path}}$ returns a tag $\mathtt{T}_1$. $\mathcal{A}$ is allowed to read and re-write the state of tag $\mathtt{T}_1$.

for $i := 1$ **to** r **do**
  $S_i \leftarrow \mathcal{O}_{\texttt{Corrupt}}(R_i)$;
end
$\texttt{T}_0 \leftarrow \mathcal{O}_{\texttt{Draw}}$(“tag at step ”$\texttt{v}_0$);
for $i := 0$ **to** $\rho - 1$ **do**
  $\texttt{v}_{\texttt{T}_0}^{i+1} \leftarrow \mathcal{O}_{\texttt{Step}}(\texttt{T}_0)$;
  $s_{\texttt{T}_0}^i$:=READ($\texttt{T}_0$);
  WRITE($\texttt{T}_0, s_{\texttt{T}_0}^{\prime i}$);
  for $j := 1$ **to** s **do**
    $T_{(i,j)} \leftarrow \mathcal{O}_{\texttt{Draw}}$(“tag's next step is $\texttt{v}_{\texttt{T}_0}^{i+1}$”));
    $s_{T_{(i,j)}}$:=READ($T_{(i,j)}$);
    WRITE($T_{(i,j)}, s'_{T_{(i,j)}}$);
  end
  for $j := 1$ **to** t **do**
    $T'_{(i,j)} \leftarrow \mathcal{O}_{\texttt{Draw}}(\texttt{c}_{(i,j)})$;
    $\texttt{v}_{T'_{(i,j)}} \leftarrow \mathcal{O}_{\texttt{Step}}(T'_{(i,j)})$;
    $s_{T'_{(i,j)}}$:=READ($T'_{(i,j)}$);
    WRITE($T'_{(i,j)}, s'_{T'_{(i,j)}}$);
  end
  ITERATESUPPLYCHAIN;
  for $j := 1$ **to** s **do**
    READ($T_{(i,j)}$);
  end
  for $j := 1$ **to** t **do**
    READ($T'_{(i,j)}$);
  end
end

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

$\texttt{T}_1 \leftarrow \mathcal{O}_{\texttt{Path}}(\texttt{P}_{\texttt{T}_0})$;
$s_{\texttt{T}_1}^k$:=READ($\texttt{T}_1$);
WRITE($\texttt{T}_1, s_{\texttt{T}_1}^{\prime k}$);
ITERATESUPPLYCHAIN;
$s_{\texttt{T}_1}^{k+1}$:=READ($\texttt{T}_1$);
OUTPUT b;

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

$\mathcal{A}$ iterates the supply chain, and reads the state stored into tag $\texttt{T}_1$.

Let $\texttt{v}_{\texttt{T}_1}$ denote the step that tag T_1 went through during the challenge phase. $\mathcal{A}$'s goal is to decide whether the step $\texttt{v}_{\texttt{T}_1} \in \texttt{P}_{\texttt{T}_0}$ or not. If $\texttt{v}_{\texttt{T}_1} \in \texttt{P}_{\texttt{T}_0}$, then $\mathcal{A}$ outputs $b = 1$; otherwise, $\mathcal{A}$ outputs $b = 0$.

The adversary $\mathcal{A}$ is successful, if **i.)** his guess of bit b is correct, and if **ii.)** the step $\texttt{v}_{\texttt{T}_1}$ is not corrupted by $\mathcal{A}$.

DEFINITION 5 (STEP UNLINKABILITY). CHECKER *provides step unlinkability*$\Leftrightarrow$ *For adversary $\mathcal{A}$, inequality Pr($\mathcal{A}$ is successful) $\leq \frac{1}{2} + \epsilon$ holds, where ϵ is negligible.*

A.2 Tag unlinkability and step unlinkability

The following theorem states that if CHECKER ensures tag unlinkability, it will as well ensure step unlinkability.

THEOREM 3. *If* CHECKER *ensures tag unlinkability, then it also ensures step unlinkability.*

PROOF. Assume there is an adversary $\mathcal{A}$ who breaks the step unlinkability with a non negligible advantage ϵ. We show that there is an adversary $\mathcal{A}'$ who uses $\mathcal{A}$ to break the tag unlinkability as defined in Section 3.2 with a non negligible advantage ϵ'.

Proof overview. In the proof below, we show that if adversary $\mathcal{A}$ has a non-negligible advantage ϵ in breaking step unlinkability, then $\mathcal{A}'$ can construct a statistical distinguisher that tells tags $\texttt{T}_0$ and $\texttt{T}_1$ apart in the tag unlinkability experiment with a non negligible advantage ϵ'.

In a nutshell, adversary $\mathcal{A}'$ supplies adversary $\mathcal{A}$ in the learning phase of step unlinkability with the first challenge tag $\texttt{T}_0$ that he receives in the tag unlinkability experiment.

Then, at the beginning of the challenge phase of the step unlinkability experiment, $\mathcal{A}'$ provides $\mathcal{A}$ with the second challenge tag $\texttt{T}_1$ of tag unlinkability.

Finally, at the end of the challenge phase of step unlinkability, $\mathcal{A}'$ replaces tag $\texttt{T}_1$ by tag $\texttt{T}_b$ that was returned by $\mathcal{O}_{\texttt{Flip}}$.

If $b = 1$, then $\mathcal{A}$ breaks the step unlinkability of CHECKER with a non-negligible advantage ϵ. Otherwise, $\mathcal{A}$'s advantage is negligible in breaking step unlinkability.

Now, the statistical distinguisher works as follows: whenever $\mathcal{A}$ outputs a correct guess for the step unlinkability experiment, then $\mathcal{A}'$ outputs $b = 1$; otherwise, $\mathcal{A}'$ outputs $b = 0$.

Construction. $\mathcal{A}'$ simulates CHECKER to adversary $\mathcal{A}$ whose goal is to break step unlinkability.

I.) In the learning phase of step unlinkability:

- Whenever $\mathcal{A}$ wants to corrupt a reader R_i, $\mathcal{A}'$ makes a query to the oracle $\mathcal{O}_{\texttt{Corrupt}}$ with R_i's identity. $\mathcal{O}_{\texttt{Corrupt}}$ returns the secrets of reader R_i to $\mathcal{A}'$, who then returns the same secrets to $\mathcal{A}$.
- When $\mathcal{A}$ queries $\mathcal{A}'$ to get a tag $\texttt{T}_0$, $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\texttt{Draw}}$. $\mathcal{O}_{\texttt{Draw}}$ returns two challenge tags $\texttt{T}_0$ and $\texttt{T}_1$ for the tag unlinkability experiment which just entered the supply chain. Then, $\mathcal{A}'$ picks for instance tag $\texttt{T}_0$ and returns $\texttt{T}_0$ to $\mathcal{A}$.
- When $\mathcal{A}$ queries $\mathcal{A}'$ to supply him with the next step of tag $\texttt{T}_0$, $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\texttt{step}}$ with tag $\texttt{T}_0$. $\mathcal{O}_{\texttt{step}}$ returns the next step $\texttt{v}_{\texttt{T}_0}^{i+1}$ of tag $\texttt{T}_0$ to $\mathcal{A}'$, who then returns $\texttt{v}_{\texttt{T}_0}^{i+1}$ to $\mathcal{A}$.
- If $\mathcal{A}$ queries $\mathcal{A}'$ for tags whose next step is $\texttt{v}_{\texttt{T}_0}^{i+1}$, then $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\texttt{Draw}}$ with the condition “tag's next step $\texttt{v}_{\texttt{T}_0}^{i+1}$”. $\mathcal{O}_{\texttt{Draw}}$ supplies $\mathcal{A}'$ with tags $T_{(i,j)}$ satisfying the condition. Finally, $\mathcal{A}'$ gives $\mathcal{A}$ the same set of tags $T_{(i,j)}$.
- If $\mathcal{A}$ queries $\mathcal{A}'$ for tags satisfying some condition $\texttt{c}_{(i,j)}$, then $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\texttt{Draw}}$ with $\texttt{c}_{(i,j)}$. $\mathcal{O}_{\texttt{Draw}}$ furnishes $\mathcal{A}'$ with tags $T'_{(i,j)}$ fulfilling the condition $\texttt{c}_{(i,j)}$. $\mathcal{A}'$ then returns the tags $T'_{(i,j)}$ to $\mathcal{A}$.
- $\mathcal{A}'$ iterates the supply chain.
- After iterating the supply chain, $\mathcal{A}'$ gives $\mathcal{A}$ the tags $T_{(i,j)}$ and $T'_{(i,j)}$ that $\mathcal{A}$ can read from.

II.) In the challenge phase of step unlinkability:

- $\mathcal{A}'$ simulates the oracle $\mathcal{O}_{\texttt{Path}}(\texttt{P}_{\texttt{T}_0})$ and prepares tag $\texttt{T}_1$ for adversary $\mathcal{A}$.
  1. $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\texttt{Step}}$ with the challenge tags $\texttt{T}_0$ and $\texttt{T}_1$. $\mathcal{A}'$ gets therefore the next step $\texttt{v}_{\texttt{T}_0}^{k+1}$ and $\texttt{v}_{\texttt{T}_1}^{k'+1}$ of tags $\texttt{T}_0$ and $\texttt{T}_1$ respectively.

     Note that if the reader associated with step $\texttt{v}_{\texttt{T}_1}^{k'+1}$ is corrupt by $\mathcal{A}$, and therewith by $\mathcal{A}'$, then $\mathcal{A}'$ stops the experiment as his attack against tag unlinkability is trivial.
  2. $\mathcal{A}'$ reads from tag $\texttt{T}_1$ and provides $\mathcal{A}$ with $\texttt{T}_1$.

- $\mathcal{A}$ can read and write into tag $\mathtt{T}_1$. Then, tag $\mathtt{T}_1$ is given back to $\mathcal{A}'$.
- The supply chain is iterated outside the range of $\mathcal{A}'$. The oracle $\mathcal{O}_{\mathtt{Flip}}$ returns tag $\mathtt{T}_b$, $b \in \{0, 1\}$.
- $\mathcal{A}'$ returns tag $\mathtt{T}_b$ to $\mathcal{A}$. Now, $\mathcal{A}$ can read from tag $\mathtt{T}_b$.
- $\mathcal{A}$ then outputs his guess for the bit b' for the step unlinkability experiment, i.e., $b' = 1$, if $\mathtt{v}_{\mathtt{T}_1}^{k'+1}$ is a step in $\mathtt{T}_0$'s path; otherwise $b' = 0$.

If $\mathtt{T}_b = \mathtt{T}_1$, then $\mathcal{A}'$'s simulation of the CHECKER system is perfect, and $\mathcal{A}$ will have a non-negligible advantage in guessing the value of the bit b'.

If $\mathtt{T}_b = \mathtt{T}_0$, then $\mathcal{A}$'s view of the step unlinkability experiment is independent of b', and thus $\mathcal{A}$'s advantage is negligible.

This therefore leads to a statistical distinguisher between tags $\mathtt{T}_0$ and $\mathtt{T}_1$. When $\mathcal{A}$ succeeds in the step unlinkability experiment, $\mathcal{A}'$ outputs $b = 1$, otherwise $\mathcal{A}'$ outputs $b = 0$.

Hence, if there is an adversary $\mathcal{A}(r, s, t, \rho, \epsilon)$ who breaks the step unlinkability of CHECKER, then there is an adversary $\mathcal{A}'(r, s + t, \rho, \epsilon)$ who breaks the tag unlinkability of CHECKER. $\square$