ROTIV: RFID Ownership Transfer with Issuer Verification

Kaoutar Elkhiyaoui Erik-Oliver Blass Refik Molva

EURECOM
2229 Route des Crêtes, BP 193
06560 Sophia Antipolis, France
{elkhiyao|blass|molva}@eurecom.fr

ABSTRACT

RFID tags travel between partner sites in a supply chain. For privacy reasons, each partner "owns" the tags present at his site, i.e., the owner is the only entity able to authenticate his tags. However, when passing tags on to the next partner in the supply chain, ownership of the old partner is "transferred" to the new partner. In this paper, we propose ROTIV, a protocol that allows for secure ownership transfer against some malicious owners. Furthermore, ROTIV offers issuer verification to prevent malicious partners from injecting fake tags not originally issued by some trusted party. As part of ownership, ROTIV provides a constant-time, privacy-preserving authentication. ROTIV's main idea is to combine an HMAC-based authentication with tag key and state updates during ownership transfer. To assure privacy, ROTIV implements tag state re-encryption techniques and key update techniques, performed on the reader. ROTIV is designed for lightweight tags – tags are only required to evaluate a hash function.

1. INTRODUCTION

Supply chain management is one of the main applications of RFID tags today. Each RFID tag is physically attached to a product to allow product tracking and inventorying. As products travel in a supply chain, their ownership is transferred from one supply chain partner to another, and so is the ownership of their corresponding RFID tags. Tag ownership in this setting is the capability that allows an owner of tag T to authenticate, access and transfer the ownership of T. Generally, the supply chain partners are reluctant into sharing their private information, therefore, each partner requires to be the only authorized entity that can interact with tags in its site. To that effect, tags and partners in the supply chain must implement secure ownership transfer protocol.

A secure ownership transfer protocol assures mutual authentication between the owner of a tag T (partner in the supply chain) and T, and prevents non-authorized parties from transferring the ownership of T without the permission of T's owner. Furthermore, to protect against counterfeiting, owners must be able to tell apart, i.e., authenticate "legitimate" tags from counterfeits. Thus, tags have to implement authentication not only to allow for ownership transfer but also to ensure their genuineness.

To ensure the privacy of tag owners, i.e., the privacy of the partners in the supply chain, a secure ownership transfer must prevent a new owner of tag T from tracing T's past interaction. Otherwise, T's new owner can infer information about T's previous owner. Also, it must prevent T's previous owner from tracing T's interactions with its current owner. Consequently, we identify two major requirements: **1)** tag backward unlinkability: ownership transfer has to prevent the previous owner of a tag from tracing a tag once he releases its ownership, see Lim and Kwon [12]. **2)** tag forward unlinkability: ownership transfer must prevent the new owner of a tag from tracing the tag's past interactions.

In addition to the basic features of ownership transfer as addressed in Fouladgar and Afifi [6], Lim and Kwon [12], Molnar et al. [13], Song [17], this paper proposes an efficient *ownership transfer* protocol that also allows a party possessing the right credentials to *verify the issuer* of a tag. A possible scenario for issuer verification is a supply chain where partners want to check that a product originates from a trusted partner. Let $\mathcal{S}$ be a supply chain consisting of four partners A, B, C, and D, whereby A and B issue products and transfer them to C, and C is required to transfer some of his products to D. However, D only accepts products originating from A. In this scenario, issuer verification is required in order to allow D to verify the origin of product received from C. The work at hand proposes a secure ownership transfer protocol that *also* achieves secure issuer verification, called ROTIV.

An efficient ownership transfer protocol calls for an efficient authentication protocol. Current RFID authentication schemes based on symmetric cryptographic primitives require at least a logarithmic cost in the number of tags, see Burmester et al. [4]. However, given that the tag databases are usually huge and that the readers performing the authentication are embedded, the tag/reader authentication should ideally take place in constant time. Previously proposed tag/reader authentication protocols that achieve constant time authentication rely on public key cryptography [11], yet RFID tags are constrained devices that most of the time cannot implement asymmetric cryptography. The authentication that is part of ROTIV achieves mutual authentication in constant time while the tag performs only *symmetric* cryptographic operations: a tag in ROTIV is only required to compute hash functions. To achieve constant time authentication, a tag T stores in addition to its *symmetric* key, a public key encryption of its identification information computed by T's owner. The public key encryption helps the owner to identify the tag T first, then the symmetric key is used to authenticate both T and its owner. In order to ensure tag unlinkability, we update the tag internal state after each successful authentication as follows: **1)** the symmetric key stored on tag T is updated using a fresh randomness transmitted by T' owner during each authentication as in [12]. **2)** the public key ciphertext is updated using re-encryption mechanisms.

Moreover, to allow the ownership transfer of tag T to a new owner $O_{(T,k+1)}$, the current owner $O_{(T,k)}$ is required to provide $O_{(T,k+1)}$ with some references that allow $O_{(T,k+1)}$ to authenticate himself to T and to update T's symmetric key.

Finally, to allow tag issuer verification by third parties, a tag T stores an encryption of the issuer signature. Provided with the correct references from T's owner, a third party verifier can verify whether the signature stored on T corresponds to a legitimate issuer or not.

In a nutshell, the major contributions of ROTIV are:

- a *provably secure* ownership transfer that ensures both tag forward unlinkability against the tag new owner and tag backward unlinkability against the tag previous owner.
- a *provably secure, privacy-preserving*, and *constant time* authentication while tags are only required to compute a hash function.
- a *provably secure* issuer verification protocol that allows prospective owners of a tag T to check the identity of the party issuing T.

2. PROBLEM STATEMENT AND ADVERSARY MODEL

Before presenting the privacy and security requirements of secure ownership transfer with issuer verification, we introduce involved entities.

2.1 Entities

- **Tags** T_i: Each tag is attached to a single item. A tag T_i has a re-writable memory representing T_i's current state $s_{(i,j)}$ at time j. Tags can compute hash function G. $\mathcal{T}$ denotes the set of legitimate tags T_i.
- **Issuer** $\mathcal{I}$: The issuer $\mathcal{I}$ initializes tags and attaches each tag T_i to a product. For each tag T_i, $\mathcal{I}$ creates a *ownership reference* $\text{ref}^O_{T_i}$ that he gives to T_i's owner. $\mathcal{I}$ writes an initial state $s_{(0,i)}$ into T_i.
- **Owner** $O_{(T_i,k)}$: Is the owner of a tag T_i at time k. $O_{(T_i,k)}$ stores a set of ownership references $\text{ref}^O_{T_i}$ that allows him to authenticate tags T_i and to transfer T_i's ownership to a new owner. $\mathbb{O}$ denotes the set of all owners $O_{(T_i,k)}$. Without loss of generality, an owner $O_{(T_i,k)}$ comprises a database $\mathcal{D}_k$ and an RFID reader R_k.
- **Verifier** $\mathcal{V}$: Before accepting the ownership of some tag T_i, any prospective owner $O_{(T_i,k+1)}$ wants to verify the identity of tag T_i's issuer, therewith becoming a verifier $\mathcal{V}$. Owner $O_{(T_i,k)}$ of T_i provides $\mathcal{V}$ with a *verification reference* $\text{ref}^V_{T_i}$ allowing $\mathcal{V}$ to verify the identity of the issuer of T_i.

2.2 RFID Ownership Transfer with Issuer Verification

We divide the application requirements of secure ownership transfer with issuer verification into four major components. While the following paragraphs only give an informal overview about requirements and only an informal overview about the assumed adversary model, sections 2.3 and 2.3.3 formalize requirements.

1.) During daily operations, current owner $O_{(T_i,k)}$ of tag T_i in the supply chain has to be able to perform a number of **mutual authentications** with T_i.

2.) Eventually, $O_{(T_i,k)}$ has to pass T_i to the next owner $O_{(T_i,k+1)}$ in the supply chain. Therefore, $O_{(T_i,k)}$ and $O_{(T_i,k+1)}$ must **exchange** the **ownership references**.

3.) Once previous owner $O_{(T_i,k)}$ releases ownership of a tag T_i, new owner $O_{(T_i,k+1)}$ must securely **update** any **secrets** stored on T_i, such that only $O_{(T_i,k+1)}$ is able to authenticate T_i and eventually pass T_i to the next owner $O_{(T_i,k+2)}$.

4.) Before accepting tag ownership, verifier $\mathcal{V}$, has to perform **issuer verification**. That is, upon receipt of T_i verification references $\text{ref}^V_{T_i}$ from T_i's current owner, $\mathcal{V}$ is able to verify whether T_i has been originally issued by $\mathcal{I}$.

Adversary Model.

The adversary model presented in this section is in accordance with previous work on secure ownership transfer, cf., Lim and Kwon [12], and with previous work on privacy model for RFID, cf., Vaudenay [18]. An adversary $\mathcal{A}$ in ROTIV is an active adversary, who in addition to being able to eavesdrop on tags' communication can tamper with tags' internal state.

$\mathcal{A}$, however does neither have access to the communication between owners during an ownership transfer, nor to the communication between a tag owner and the verifier during an issuer verification protocol, since the channel between the owners is secure and also the channel between the owner and the verifier. Therefore, $\mathcal{A}$ has only access to the interactions between tags and owners and the interactions between tags and verifiers.

We assume that the channel between $\mathcal{V}$ and T's current owner is secure. The verifier in ROTIV is assumed to be honest.

2.3 Privacy

Our application models partners in a supply chain that release or acquire tag ownership. When a partner A releases T_i's ownership to another partner B, it is very important to ensure that A is not able to trace the tag's future protocol runs: if A can trace T_i after the ownership transfer, he will be able to learn how B processes the products received from A. Also, it is important to make sure that B cannot trace the tag past interactions, otherwise, B can learn how products are processed when they are in A's site.

Along these lines, we identify two precise privacy requirements which are *tag forward unlinkability* [12] and *tag backward unlinkability* [12]. Generally speaking, forward unlinkability states that if an adversary $\mathcal{A}$ compromises the internal state of a tag T at time τ, he still cannot tell whether T has participated in protocol runs at time $t < \tau$.

On the other hand, backward unlinkability copes with an adversary $\mathcal{A}$ who, even knowing the internal state of a tag T at time τ, cannot tell whether T was involved in protocol runs that occurred at $t > \tau$.

We formalize tag *forward* unlinkability and tag *backward* unlinkability using games.

We assume that the adversary $\mathcal{A}$ has access to the following oracles.

- $\mathcal{O}_{\mathcal{T}}$ is an oracle that, when queried, randomly returns a tag T from the set of tags $\mathcal{T}$.

- $\mathcal{O}_{\text{flip}}$ is an oracle that, when provided with two tags T_0 and T_1, randomly chooses $b \in \{0, 1\}$ and returns T_b.

2.3.1 Forward unlinkability

The forward unlinkability experiment captures the capabilities of adversary $\mathcal{A}$ who is allowed to access a tag T's internal state at the *end* of its attack and who has to decide if T was already involved in previous interactions.

Our forward unlinkability experiment is indistinguishability based as proposed by Juels and Weis [9]. Adversary $\mathcal{A}(r, s, \epsilon)$ has access to tags in two phases. In the learning phase, as depicted in Algorithm 1, oracle $\mathcal{O}_{\mathcal{T}}$ provides $\mathcal{A}$ with two tag T_0 and T_1 that he can observe T_0 and T_1's interactions for a maximum of s times by calling the function ObserveInteraction(T_i). This function eavesdrop on tag T_i during mutual authentications, ownership transfer or issuer verification.

In addition to T_0 and T_1, $\mathcal{O}_{\mathcal{T}}$ gives $\mathcal{A}$ a set of r tags T'_i. $\mathcal{A}$ can read T'_i's internal state (cf., READSTATE) and modify it (cf., MODIFYSTATE) up to s times. He can as well eavesdrop on T'_i (cf., OBSERVEINTERACTION(T'_i)) for a maximum of s times.

```
T_0 ← O_T;
T_1 ← O_T;
for j := 1 to s do
    ObserveInteraction(T_0);
    ObserveInteraction(T_1);
end
for i := 1 to r do
    T'_i ← O_T;
    for j := 1 to s do
        s_(T'_i,j) := ReadState(T'_i);
        ModifyState(T'_i, s'_T'_i);
        ObserveInteraction(T'_i);
    end
end
```

Algorithm 1: $\mathcal{A}$'s forward unlinkability learning phase

In the challenge phase as depicted in Algorithm 2, T_0 and T_1 run once a mutual authentication with their respective owners (cf., RUNAUTH) *outside* the range of the adversary $\mathcal{A}$. $\mathcal{A}$ then queries oracle $\mathcal{O}_{\text{flip}}$ with the tags T_0 and T_1. $\mathcal{O}_{\text{flip}}$ selects randomly $b \in \{0, 1\}$ and returns the tag T_b. $\mathcal{A}$ can read the internal state of T_b. He can also eavesdrop T_b for a maximum of s times. $\mathcal{A}$ calls as well oracle $\mathcal{O}_{\mathcal{T}}$ that provides $\mathcal{A}$ with r tags T''_i that he can read out and tamper their internal state up to s times. He can eavesdrop on T''_i for a maximum of s times. Finally, $\mathcal{A}$ outputs his guess of the value of b.

```
RunAuth(T_0, O_(T_0,k)); // Unobserved by A.
RunAuth(T_1, O_(T_1,k)); // Unobserved by A.
T_b ← O_flip{T_0, T_1};
for j := 1 to s do
    s_(T_b,j) := ReadState(T_b);
    ObserveInteraction(T_b);
end
for i := 1 to r do
    T''_i ← O_T;
    for j := 1 to s do
        s_(T''_i,j) := ReadState(T''_i);
        ModifyState(T''_i, s'_T''_i);
        ObserveInteraction(T''_i);
    end
end
Output b;
```

Algorithm 2: $\mathcal{A}$'s forward unlinkability challenge phase

$\mathcal{A}$ is successful, if his guess of b is correct.

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

The above definition reflects a malicious new owner $O_{(T,k+1)}$ of T in the real world, who after a successful ownership transfer with previous owner $O_{(T,k)}$ at time τ gets access to T's secrets. Generally, knowing T's secrets at time τ must not allow $O_{(T,k+1)}$ to track T's interactions at time $t < \tau$.

2.3.2 *Backward unlinkability*

The backward unlinkability experiment captures the capabilities of adversary $\mathcal{A}$ who is allowed to access a tag T's internal state at the *beginning* of his attack and has to tell if T is involved in future protocol transactions.

In the learning phase, cf., Algorithm 3, oracle $\mathcal{O}_{\mathcal{T}}$ provides $\mathcal{A}(r, s, \epsilon)$ with two tag T_0 and T_1 that he can read their internal state, he can also eavesdrop T_0 and T_1 for a maximum of s times. Besides T_0 and T_1, $\mathcal{O}_{\mathcal{T}}$ gives $\mathcal{A}$ a set of r tags T'_i that he can read and modify their internal state for a maximum of s times. He can also eavesdrop on T'_i for a maximum of s times. Note that unlike forward unlinkability, $\mathcal{A}$ can read the internal state of T_0 and T_1 in the learning phase of the backward unlinkability experiment, but not in the challenge phase.

```
T_0 ← O_T;
T_1 ← O_T;
for j := 1 to s do
    s_(T_0,j) := ReadState(T_0);
    ObserveInteraction(T_0);
    s_(T_1,j) := ReadState(T_1);
    ObserveInteraction(T_1);
end
for i := 1 to r do
    T'_i ← O_T;
    for j := 1 to s do
        s_(T'_i,j) := ReadState(T'_i);
        ModifyState(T'_i, s'_T'_i);
        ObserveInteraction(T'_i);
    end
end
```

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

In the challenge phase as depicted in Algorithm 4, T_0 and T_1 run a mutual authentication with their respective owners *outside* the range of the adversary $\mathcal{A}$. $\mathcal{A}$ provides oracle $\mathcal{O}_{\text{flip}}$ with the tags T_0 and T_1. $\mathcal{O}_{\text{flip}}$ chooses randomly $b \in \{0, 1\}$ and returns the tag T_b. Unlike the challenge phase of the forward unlinkability $\mathcal{A}$ is not allowed to read the internal state of T_b, he is only allowed to eavesdrop on T_b for a maximum of s times.

$\mathcal{A}$ queries also oracle $\mathcal{O}_{\mathcal{T}}$ that provides $\mathcal{A}$ with s tags T''_i that he can read, modify their internal state and eavesdrop on for a maximum of s times. Finally, $\mathcal{A}$ outputs his guess of the value of b.

```
RunAuth(T_0, O_(T_0,k)); // Unobserved by A.
RunAuth(T_1, O_(T_1,k)); // Unobserved by A.
T_b ← O_flip{T_0, T_1};
for j := 1 to s do
    ObserveInteraction(T_b);
end
for i := 1 to r do
    T''_i ← O_T;
    for j := 1 to s do
        s_(T''_i,j) := ReadState(T''_i);
        ModifyState(T''_i, s'_T''_i);
        ObserveInteraction(T''_i);
    end
end
Output b;
```

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

$\mathcal{A}$ is successful, if his guess of b is correct.

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

In the real world, this adversary $\mathcal{A}$ reflects previous owner $O_{(T,k)}$ who releases T's ownership at time τ to a new owner $O_{(T,k+1)}$. The knowledge of T's secrets at the time of ownership transfer τ must not allow $O_{(T,k)}$ to trace T's interaction that occur at time $t > \tau$.

Discussion: In scenarios where mutual authentication is required, the notion of *backward unlinkability* has been proven to be unachievable without tag performing public key cryptography operations [14]. In order to achieve at least a slightly weaker notion of backward unlinkability, as targeted in this paper, we add the assumption that an adversary $\mathcal{A}$ *cannot continuously* monitor the tag after accessing tags' secrets. This has been previously suggested by, e.g., Lim and Kwon [12]. That is, there is at least one communication between the tag and its owner that is unobserved by $\mathcal{A}$. Moreover, to achieve a constant time authentication while the tag is only required to compute a hash function, we assume that there is at least one unobserved communication between the tag and the owner before $\mathcal{A}$ accesses tags' secrets, as proposed by Ateniese et al. [1], Golle et al. [8].

2.3.3 Security

In the following, we discuss the security requirements for ROTIV. As ROTIV consists of two main protocols, ownership transfer protocol and the issuer verification protocol, we provide the security requirements for each protocol separately.

The adversary $\mathcal{A}$ in this section is a non-narrow destructive adversary, see Vaudenay [18].

2.3.4 Ownership transfer

A secure ownership transfer must provide the following properties:

1) Mutual authentication: A secure ownership transfer protocol must ensure that, when a tag T runs a successful mutual authentication with owner $\mathcal{O}$, this implies that $\mathcal{O}$ is T's current owner with high probability. Also, when an owner $\mathcal{O}$ runs a successful mutual authentication with a tag T, it yields that T is actually owned by $\mathcal{O}$ with high probability.

We define an authentication game in accordance with Lim and Kwon [12], Vaudenay [18] and Paise and Vaudenay [14]. This game proceeds in two phases. During the learning phase as depicted in Algorithm 5, an adversary $\mathcal{A}(r, \epsilon)$ is provided with a challenge tag T_c from oracle $\mathcal{O}_\mathcal{T}$. $\mathcal{A}$ is not allowed to read the internal state of T_c. $\mathcal{A}$ is allowed to eavesdrop on r mutual authentications between T_c and its owner $O_{(T_c,k)}$, cf., RUNAUTH(T_c, $O_{(T_c,k)}$). He can also alter r authentications by modifying the messages exchanged between T_c and its owner $O_{(T_c,k)}$, cf., ALTERAUTH(T_c, $O_{(T_c,k)}$). $\mathcal{A}$ is allowed as well to start r authentications with T_c while impersonating $O_{(T_c,k)}$, (cf., RUNAUTH(T_c, $\mathcal{A}$)). Also he can start r authentications with $O_{(T_c,k)}$ while impersonating T_c, cf., RUNAUTH($\mathcal{A}$, $O_{(T_c,k)}$)

```
T_c ← O_T;
for i = 1 to r do
    RUNAUTH(T_c, O_(T_c,k));
end
for i = 1 to r do
    ALTERAUTH(T_c, O_(T_c,k));
end
for i = 1 to r do
    RUNAUTH(T_c, A);
end
for i = 1 to r do
    RUNAUTH(A, O_(T_c,k));
end
```
Algorithm 5: $\mathcal{A}$'s authentication learning phase

$\mathcal{A}$'s goal in the challenge phase is **either** to run a successful mutual authentication with T_c, i.e., $\mathcal{A}$ succeeds in impersonating $O_{(T_c,k)}$, **or** to run a successful mutual authentication with $O_{(T_c,k)}$, i.e., $\mathcal{A}$ succeeds in impersonating T_c.

In the challenge phase as depicted in Algorithm 6, $\mathcal{A}(r, \epsilon)$ interacts with T_c and initiates an authentication protocol run to impersonate $O_{(T_c,k)}$, cf., RUNAUTH($T_c, \mathcal{A}$). At the end of the authentication, T_c outputs a bit b_{T_c}, $b_{T_c} = 1$ if the authentication with $\mathcal{A}$ was successful, and $b_{T_c} = 0$ otherwise.

$\mathcal{A}$ can interact as well with $O_{(T_c,k)}$ and initiates an authentication protocol run to impersonate T_c, cf., RUNAUTH($\mathcal{A}, O_{(T_c,k)}$). At the end of this authentication, $O_{(T_c,k)}$ outputs a bit $b_{O_{(T_c,k)}} = 1$, if the authentication was successful, $b_{O_{(T_c,k)}} = 0$ otherwise.

```
RUNAUTH(T_c, A);
T_c outputs b_T_c;
RUNAUTH(A, O_(T_c,k));
O_(T_c,k) outputs b_O_(T_c,k);
```
Algorithm 6: $\mathcal{A}$'s authentication challenge phase

$\mathcal{A}$ is successful if, $b_{T_c} = 1$ or $b_{O_{(T_c,k)}} = 1$.

DEFINITION 3 (AUTHENTICATION). *ROTIV is secure with regard to authentication $\Leftrightarrow$ For any adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ is successful$) \leq \epsilon$ holds, where ϵ is negligible.*

This definition captures the capabilities of an advesrary $\mathcal{A}$ who does not have access to tag T's internal state and who wants to either impersonate T or T's owner.

2) Exclusive ownership: An adversary $\mathcal{A}$'s goal is to transfer the ownership of a tag T without having T's *ownership* references noted ref_T^O. The exclusive ownership will ensure that only the owner of tag T can transfer T's ownership and no one else. The exclusive ownership ensures that even if an adversary $\mathcal{A}$ who does not have T's ownership references cannot cannot transfer the ownership of T, unless he rewrites the content of T.

To formalize the exclusive ownership, we define these additional oracles:

-$\mathcal{O}_{\text{own}}$ when queried with a tag T, returns T's ownership references ref_T^O from some owner O.

-$\mathcal{O}_{\mathbb{O}}$ when queried, returns a randomly selected owner O from the set of legitimate owners $\mathbb{O}$.

In the learning phase as shown in Algorithm 7, the oracle $\mathcal{O}_\mathcal{T}$ provides $\mathcal{A}(r, s, \epsilon)$ with r tags T_i. $\mathcal{A}$ then queries the oracle $\mathcal{O}_{\text{own}}$ and gets the ownership references $\text{ref}_{T_i}^O$ of tags T_i. $\mathcal{A}$ can read and modify T_i's internal state. Given the ownership references of tag T_i, $\mathcal{A}$ can run s successful mutual authentications with T_i, cf., RUNAUTH(T_i, $\mathcal{A}$), s issuer verification for T_i with $\mathcal{V}$, cf., VERIFY($T_i, \mathcal{A}, \mathcal{V}$), and s ownership transfer for tag T_i with owner O_i selected randomly from the set of owners, cf., TRANSFEROWNERSHIP ($T_i, \mathcal{A}, O_i$).

In the challenge phase, cf., Algorithm 8, the oracle $\mathcal{O}_\mathcal{T}$ provides $\mathcal{A}(r, s, \epsilon)$ with a challenge tag T_c for which $\mathcal{A}$ did not query the oracle $\mathcal{O}_{\text{own}}$. He can as well read T_c's internal state, eavesdrop on T_c's up to s times, cf., OBSERVEINTERACTION(T_c). However, $\mathcal{A}$ is not allowed to alter T_c's internal state.

for $i := 1$ **to** r **do**
  $T_i \leftarrow \mathcal{O}_{\mathcal{T}}$;
  $\mathrm{ref}^{O}_{T_i} \leftarrow \mathcal{O}_{\mathrm{own}}$;
  for $j := 1$ **to** s **do**
    $s_{(T_i,j)} :=$ READSTATE(T_i);
    MODIFYSTATE(T_i, s'_{T_i});
    RUNAUTH($T_i, \mathcal{A}$);
    VERIFY($T_i, \mathcal{A}, \mathcal{V}$);
    $O_i \leftarrow \mathcal{O}_{\mathbb{O}}$;
    TRANSFEROWNERSHIP($T_i, \mathcal{A}, O_i$);
  end
end

Algorithm 7: $\mathcal{A}$'s exclusive ownership learning phase

At the end of the challenge phase, $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathbb{O}}$. $\mathcal{O}_{\mathbb{O}}$ returns a challenge owner O_c. $\mathcal{A}$ then, runs an ownership transfer protocol for T_c with an owner O_c, cf., TRANSFEROWNERSHIP(T_c, $\mathcal{A}$, O_c). O_c outputs a bit $b = 1$, if the ownership transfer was successful, and $b = 0$ otherwise.

$T_c \leftarrow \mathcal{O}_{\mathcal{T}}$;
for $i := 1$ **to** s **do**
  $s^{i}_{T_c} :=$ READSTATE(T_c);
  OBSERVEINTERACTION(T_c);
end
$O_c \leftarrow \mathcal{O}_{\mathbb{O}}$;
TRANSFEROWNERSHIP($T_c, \mathcal{A}, O_c$);
O_c outputs b;

Algorithm 8: $\mathcal{A}$'s exclusive ownership challenge phase

$\mathcal{A}$ is successful, if $b = 1$.

DEFINITION 4 (EXCLUSIVE OWNERSHIP). *ROTIV provides exclusive ownership $\Leftrightarrow$ For any adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ is successful$) \leq \epsilon$ holds, where ϵ is negligible.*

2.3.5 Issuer verification

The second security requirement that will be discussed below, is issuer verification security. The issuer verification is secure if, when verifier $\mathcal{V}$ outputs that T's issuer is $\mathcal{I}$, it implies that $\mathcal{I}$ is T's issuer with high probability.

An adversary $\mathcal{A}$'s goal is to run an issuer verification protocol with $\mathcal{V}$ for tag T that was not issued by $\mathcal{I}$, and still $\mathcal{V}$ outputs that $\mathcal{I}$ is the issuer of T.

In the learning phase, $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathcal{T}}$ that provides $\mathcal{A}$ with r random tags T_i. $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathrm{own}}$ with tags T_i and gets T_i's ownership references. $\mathcal{A}$ is allowed to read and modify T_i's internal state up to s times. Given the ownership reference for tag T_i, $\mathcal{A}$ can run s mutual authentications between tag T_i, cf., RUNAUTH($T_c, \mathcal{A}$). The adversary can also run s issuer verification protocol for tag T_i with the verifier $\mathcal{V}$, cf., VERIFY($T_i, \mathcal{A}, \mathcal{V}$) and to transfer T_i's ownership to an owner O_i select from the set of owner $\mathbb{O}$.

In the challenge phase, $\mathcal{A}$ creates a tag $T_c \notin \mathcal{T}$ and write some state s'_{T_c} in it.

Then, $\mathcal{A}$ starts a verification protocol for tag T_c with the verifier $\mathcal{V}$, cf., VERIFY ($T_c, \mathcal{A}, \mathcal{V}$).

Finally, $\mathcal{V}$ outputs a bit $b = 1$, if the issuer verification protocol outputs $\mathcal{I}$, and $b = 0$ otherwise.

$\mathcal{A}$ is successful, if $b = 1$ and s'_{T_c} does not correspond to a state of tag T_i that was provided to $\mathcal{A}$ in the learning phase.

for $i := 1$ **to** r **do**
  $T_i \leftarrow \mathcal{O}_{\mathcal{T}}$;
  $\mathrm{ref}^{O}_{T_i} \leftarrow \mathcal{O}_{\mathrm{own}}$;
  for $j := 1$ **to** s **do**
    $s_{(T_i,j)} :=$ READSTATE(T_i);
    MODIFYSTATE(T_i, s'_{T_i});
    RUNAUTH($T_i, \mathcal{A}$);
    VERIFY(T_i,$\mathcal{A}$, $\mathcal{V}$);
    $O_i \leftarrow \mathcal{O}_{\mathbb{O}}$;
    TRANSFEROWNERSHIP($T_i, \mathcal{A}, O_i$);
  end
end

Algorithm 9: $\mathcal{A}$'s issuer verification security learning phase

CREATETAG T_c;
MODIFYSTATE(T_c, s'_{T_c});
VERIFY (T_c, $\mathcal{A}$, $\mathcal{V}$);
$\mathcal{V}$ outputs b;

Algorithm 10: $\mathcal{A}$'s issuer verification security challenge phase

DEFINITION 5 (ISSUER VERIFICATION SECURITY). *ROTIV is secure with regard to issuer verification $\Leftrightarrow$ For any adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ is successful$) \leq \epsilon$ holds, where ϵ is negligible.*

In real world, a secure issuer verification will prevent a partner in the supply chain from injecting tags that were not issued by a legitimate/trusted party.

3. PROTOCOL DESCRIPTION

ROTIV takes place in DDH-hard subgroups of elliptic curves that support bilinear pairings, cf., Ateniese et al. [1, 2], Ballard et al. [3].

3.1 Bilinear pairing

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 prime order q. Pairing e: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear pairing if:

1. e is *bilinear*: $\forall\, a, b \in \mathbb{Z}_q, g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
2. e is *computable*: there is an efficient algorithm to compute $e(g_1, g_2)$ for any $(g_1, g_2) \in \mathbb{G}_1 \times \mathbb{G}_2$;
3. e is *non-degenerate*: if g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$, then $e(g_1, g_2)$ is a generator $\mathbb{G}_T$.

In ROTIV, we use bilinear groups where DDH is intractable, i.e., groups where the symmetric external Diffie-Hellman (SXDH) assumption, see Ateniese et al. [1, 2], Ballard et al. [3], holds. Such groups can be chosen as specific subgroups of MNT curves. Furthermore, results by Galbraith et al. [7] indicate the high efficiency of this pairing.

DEFINITION 6 (SXDH ASSUMPTION). *The SXDH 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 intractable in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

For our pairing, we also assume Bilinear CDH.

DEFINITION 7 (BCDH ASSUMPTION). *Let g_1 be a generator of $\mathbb{G}_1$ and g_2 be a generator of $\mathbb{G}_2$. We say that the BCDH assumption holds if, given $g_1, g_1^a, g_1^b, g_1^c \in \mathbb{G}_1$ and $g_2, g_2^a, g_2^b \in \mathbb{G}_2$ for random $a, b, c \in \mathbb{F}_q$, the probability to compute $e(g_1, g_2)^{abc}$ is negligible.*

3.2 ROTIV

Overview: In ROTIV, a tag T_i stores a state $s_{(i,j)} = (k_{(i,j)}, c_{(i,j)})$, where $k_{(i,j)}$ is a key shared with the owner of T_i, and $c_{(i,j)}$ an Elgamal encryption of T_i's identification information.

When an owner $O_{(T_i,k)}$ starts a mutual authentication with T_i, T_i replies with $c_{(i,j)}$ along with an HMAC computed using his secret key $k_{(i,j)}$. $O_{(T_i,k)}$ uses the Elgamal ciphertext $c_{(i,j)}$ to identify the tag. To do so, $O_{(T_i,k)}$ uses his secret key to decrypt $c_{(i,j)}$. After the decryption, $O_{(T_i,k)}$ checks if the resulting plaintext is in his database $\mathcal{D}_k$. If so, $O_{(T_i,k)}$ looks up the symmetric key of tag T_i in his database and verifies the HMAC sent by T_i. In this manner ROTIV allows for mutual authentication with tag T_i in constant time, while the tag is only required to compute a symmetric primitive, i.e., HMAC.

To allow for ownership transfer of tag T_i, the current owner $O_{(T_i,k)}$ of T_i provides $O_{(T_i,k+1)}$ with $\text{ref}_{T_i}^O$ that will be used by $O_{(T_i,k+1)}$ to authenticate himself to T_i and to update T_i's state.

In order to ensure T_i's forward and backward privacy, the owner $O_{(T_i,k)}$ of T_i is required to update the ciphertext stored on T_i in every authentication he runs with T_i, using re-encryption mechanisms. Moreoever, T_i is required as well to update its key $k_{(i,j)}$ after each successful authentication.

Finally, for secure issuer verification, ciphertext $c_{(i,j)}$ stored on T_i will contain a signature of $\mathcal{I}$ on the identifier of T_i. When a verifier $\mathcal{V}$ wants to start a secure issuer verification for a tag T_i, he reads the ciphertext stored in T_i. Then, T_i's owner, $O_{(T_i,k)}$ sends T_i's identifier and a trapdoor information noted $\text{ref}_{T_i}^V$ to $\mathcal{V}$. This will allow $\mathcal{V}$ to verify the signature stored in T_i.

A ROTIV system comprises m owners $O_{(T_i,k)}$ and n tags T_i. Each tag T_i can evaluate a cryptograpgic hash function G to compute HMAC. An HMAC with key k, a message m is defined in ROTIV as $\text{HMAC}_k(m) = G(k \oplus \text{opad}||G(k \oplus \text{ipad}||m))$, where $||$ is concatenation. For more details about opad and ipad see Krawczyk et al. [10]. The HMAC is used to authenticate T_i and T_i's owner, and to update the symmetric key after each successful authentication.

• **Setup:** The issuer $\mathcal{I}$ outputs $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, where $\mathbb{G}_1$, $\mathbb{G}_T$ are subgroups of prime order q, g_1 and g_2 are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear pairing. The issuer chooses $x \in \mathbb{Z}_q^*$ and computes the pair (g_1^x, g_2^x). $\mathcal{I}$'s secret key is $sk = (x, g_1^x)$ and his public key is $pk = g_2^x$.

For each owner $O_{(T_i,k)}$ $\mathcal{I}$ randomly selects $\alpha_k \in \mathbb{Z}_q^*$ and computes the pair $(g_1^{\alpha_k}, g_2^{\alpha_k})$. The system provides each owner $O_{(T_i,k)}$ with his secret key $sk = \alpha_k$ and his public key $pk = (g_1^{\alpha_k^2}, g_2^{\alpha_k})$. All owners know each other's public key.

• **Tag Initialization:** The issuer $\mathcal{I}$ initializes a tag T_i owned by $O_{(T_i,k)}$. $\mathcal{I}$ picks a random number $t_i \in \mathbb{F}_q$. Using a cryptographic hash function $H : \mathbb{F}_q \rightarrow \mathbb{G}_1$, $\mathcal{I}$ computes $h_i = H(t_i) \in \mathbb{G}_1$. Then, $\mathcal{I}$ computes $u_{(i,0)} = 1$ and $v_{(i,0)} = h_i^x$. Finally, $\mathcal{I}$ chooses randomly a key $k_{(i,0)} \in \mathbb{F}_q$. Tag T_i stores: $s_{(i,0)} = (k_{(i,0)}, c_{(i,0)})$, where $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)})$.

$\mathcal{I}$ provides $O_{(T_i,k)}$ with tag T_i, $\text{ref}_{T_i}^O = (k_i^{\text{old}}, k_i^{\text{new}}, x_i, y_i) = (k_{(i,0)}, k_{(i,0)}, t_i, h_i^x)$.

Before accepting the tag, $O_{(T_i,k)}$ reads T_i and checks if the ownership references verify the following equation:

$$e(H(x_i), g_2^x) = e(y_i, g_2)$$

This equation implies that T_i is actually issued by $\mathcal{I}$, that is $y_i = H(x_i)^x$.

The owner $O_{(T_i,k)}$ adds an entry E_{T_i} for tag T_i in his database $\mathcal{D}_k$: $E_{T_i} = (y_i, \text{ref}_{T_i}^O)$. y_i acts as the index of T_i in $O_{(T_i,k)}$'s database $\mathcal{D}_k$.

Once the owner $O_{(T_i,k)}$ accepts the tag, he overwrites its content. He chooses randomly $r_{(i,1)} \in \mathbb{F}_q$ and computes an Elgamal encryption of y_i using his public key $g_1^{\alpha^2}$: $c_{(i,1)} = (u_i^1, v_{(i,1)}) = (g_1^{r_{(i,1)}}, y_i g_1^{\alpha_k^2 r_{(i,1)}})$, see El Gamal [5]. Therefore,

$$s_{(i,1)} = (k_{(i,1)} = k_{(i,0)}, c_{(i,1)}).$$

3.2.1 *Authentication protocol*

To authenticate a tag T_i, the owner $O_{(T_i,k)}$ decrypts the ciphertext $c_{(i,j)} = (u_{(i,j)}, v_{(i,j)})$ and gets y_i. Using y_i, $O_{(T_i,k)}$ identifies T_i and starts a hash-based mutual authentication. to compute $y_i(g_1^{r_{(i,j)}}, \delta_i g_1^{\alpha_k^2 r_{(i,j)}})$. If the mutual authentication succeeds, both the owner $O_{(T_i,k)}$ and the tag T_i update their keys.

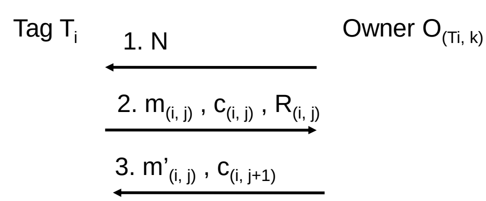

Figure 1: Authentication in ROTIV

1) To start an authentication with tag T_i, the owner $O_{(T_i,k)}$ sends a random nonce N to T_i as depicted in figure 3.2.1.

2) T_i generates a random number $R_{(i,j)} \in \mathbb{F}_q$. Using his secret key $k_{(i,j)}$, T_i computes:

$$m_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(N, R_{(i,j)}, c_{(i,j)}) \quad (1)$$

T_i finally replies with $(R_{(i,j)}, c_{(i,j)} = (u_{(i,j)}, v_{(i,j)}), m_{(i,j)})$. To generate $R_{(i,j)}$, T_i can either use physical noise to extract $R_{(i,j)}$, or use a counter count_i and computes $R_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(\text{count}_i)$.

3) Upon receiving T_i's reply, the owner $O_{(T_i,k)}$ computes:

$$y_i = \frac{v_{(i,j)}}{(u_{(i,j)})^{\alpha_k^2}}$$

$O_{(T_i,k)}$ checks if $y_i \in \mathcal{D}_k$. If not, $O_{(T_i,k)}$ aborts authentication. Otherwise, $O_{(T_i,k)}$ looks up T_i's ownership references $\text{ref}_{T_i}^O = (k_i^{\text{old}}, k_i^{\text{new}}, t_i, h_i^x)$ in $\mathcal{D}_k$ and checks if:

$$m_{(i,j)} = \text{HMAC}_{k_i^{\text{new}}}(N, R_{(i,j)}, c_{(i,j)}) \ \textbf{OR}$$
$$m_{(i,j)} = \text{HMAC}_{k_i^{\text{old}}}(N, R_{(i,j)}, c_{(i,j)})$$

If not, $O_{(T_i,k)}$ aborts authentication. If $\text{HMAC}_{k_i^{\text{old}}}(N, R_{(i,j)}, c_{(i,j)}) = m_{(i,j)}$ then $k_{(i,j)} = k_i^{\text{old}}$, otherwise $k_{(i,j)} = k_i^{\text{new}}$.

$O_{(T_i,k)}$ chooses a new random number $r_{(i,j+1)} \in \mathbb{F}_q^*$ and computes:

$$c_{(i,j+1)} = (u_{(i,j+1)}, v_{(i,j+1)}) = (g_1^{r_{(i,j+1)}}, h_i^x g_1^{\alpha_k^2 r_{(i,j+1)}}) \quad (2)$$
$$m'_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(R_{(i,j)}, c_{(i,j+1)}) \quad (3)$$

$O_{(T_i,k)}$ sends $c_{(i,j+1)}$ and $m'_{(i,j)}$ to T_i. Finally, $O_{(T_i,k)}$ updates the symmetric keys k_i^{old} and k_i^{new} in his database $\mathcal{D}_k$:

$$(k_i^{\text{old}}, k_i^{\text{new}}) = (k_{(i,j)}, G(k_{(i,j)}, N)) \quad (4)$$

4) Once T_i receives $m'_{(i,j)}$ and $c_{(i,j+1)}$, and checks if $m'_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(R_{(i,j)}, c_{(i,j+1)})$. If not T_i aborts authentication, otherwise, T_i updates its state $s_{(i,j)}$ to $s_{(i,j+1)}$. To do so, T_i computes its new key $k_{(i,j+1)}$.

$$k_{(i,j+1)} = G(k_{(i,j)}, N) \quad (5)$$

T_i updates its state $s_{(i,j+1)} = (k_{(i,j+1)}, c_{(i,j+1)})$.

3.2.2 Issuer verification protocol

In order to verify whether a tag T_i owned by $O_{(T_i,k)}$ is issued by $\mathcal{I}$, $\mathcal{V}$ proceeds as follows:

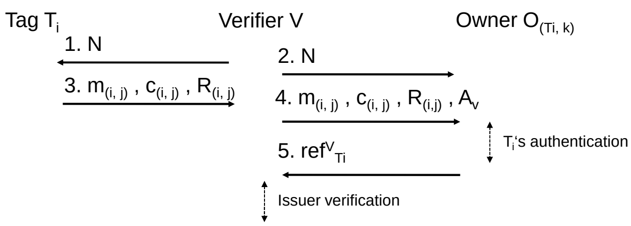

Figure 2: Issuer verification in ROTIV

1) $\mathcal{V}$ sends a Nonce N to T_i, as if he is starting a mutual authentication with T_i, as depicted in figure 2.

2) T_i replies with $c_{(i,j)} = (u_{(i,j)}, v_{(i,j)}) = (g_1^{r(i,j)}, h_i^x g_1^{\alpha_k^2 r(i,j)})$, a hash $m_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(N, R_{(i,j)}, c_{(i,j)})$ and a random number $R_{(i,j)}$.

3) $\mathcal{V}$ chooses a random number $r_v \in \mathbb{F}_q^*$, he computes $A_v = (u_{(i,j)})^{r_v} = g_1^{r(i,j) r_v}$. $\mathcal{V}$ forwards N, $R_{(i,j)}$,$c_{(i,j)}$, $m_{(i,j)}$ along with A_v to $O_{(T_i,k)}$.

4) Upon receiving the tuple $(N, R_{(i,j)}, c_{(i,j)}, m_{(i,j)}, A_v)$, $O_{(T_i,k)}$ identifies and authenticates T_i. If $O_{(T_i,k)}$ is not willing to run the verification protocol for T_i he aborts the verification. Otherwise, $O_{(T_i,k)}$ computes:

$$\text{ref}_{T_i}^V = (A_{(i,j)}, B_{(i,j)}, C_{(i,j)}) = (t_i, H(t_i)^x, A_v^{\alpha_k})$$

Finally, $O_{(T_i,k)}$ sends $\text{ref}_{T_i}^V = (A_{(i,j)}, B_{(i,j)}, C_{(i,j)})$ to $\mathcal{V}$.

5) Provided with the verification references $\text{ref}_{T_i}^V$, $\mathcal{V}$ checks whether the following equations hold:

$$e(H(A_{(i,j)}), g_2^x) = e(B_{(i,j)}, g_2) \quad (6)$$
$$e(C_{(i,j)}, g_2) = e(A_v, g_2^{\alpha_k}) \quad (7)$$

Equation (6) verifies whether $B_{(i,j)} = H(A_{(i,j)})^x$, i.e., whether $B_{(i,j)}$ is the signature of $A_{(i,j)}$ by issuer $\mathcal{I}$. Equation (7) checks whether $C_{(i,j)} = A_v^{\alpha_k}$.

Finally, $\mathcal{V}$ checks whether $c_{(i,j)}$ is the encryption of $B_{(i,j)}$ with the public key $g_1^{\alpha_k^2}$. To do so, $\mathcal{V}$ checks if the following equation holds:

$$e(v_{(i,j)}, g_2)^{r_v} = e(B_{(i,j)}, g_2)^{r_v} e(C_{(i,j)}, g_2^{\alpha_k}) \quad (8)$$

Note that if $c_{(i,j)}$ is the encryption of $B_{(i,j)}$ with the public key $g_1^{\alpha_k^2}$, we have: $c_{(i,j)} = (u_{(i,j)}, v_{(i,j)}) = (g_1^{r(i,j)}, B_{(i,j)} g_1^{\alpha_k^2 r(i,j)})$.

Therefore,

$$\begin{aligned} e(v_{(i,j)}, g_2)^{r_v} &= e(B_{(i,j)}, g_2)^{r_v} e(g_1^{\alpha_k^2 r(i,j)}, g_2)^{r_v} \\ &= e(B_{(i,j)}, g_2)^{r_v} e(g_1^{r_v r(i,j)}, g_2^{\alpha_k^2}) \\ &= e(B_{(i,j)}, g_2)^{r_v} e(A_v, g_2^{\alpha_k^2}) \\ &= e(B_{(i,j)}^x, g_2)^{r_v} e(A_v^{\alpha_k}, g_2^{\alpha_k}) \\ &= e(B_{(i,j)}, g_2^x)^{r_v} e(C_{(i,j)}, g_2^{\alpha_k}) \end{aligned}$$

If all the equations hold, $\mathcal{V}$ outputs $b = 1$ meaning that $\mathcal{I}$ is T_i's issuer. Otherwise, $\mathcal{V}$ outputs $b = 0$ meaning that $\mathcal{I}$ is not the issuer of T_i.

3.2.3 Ownership transfer protocol

The setup of the ownership transfer in ROTIV consists of a previous owner $O_{(T_i,k)}$, a new owner $O_{(T_i,k+1)}$ and a tag T_i as shown in figure 3. The ownership transfer consists of a mutual authentication between T_i and $O_{(T_i,k+1)}$, and an exchange of ownership references $\text{ref}_{T_i}^O$ between $O_{(T_i,k)}$ and $O_{(T_i,k+1)}$. These ownership references $\text{ref}_{T_i}^O$ are what allows for for $O_{(T_i,k+1)}$ authentication.

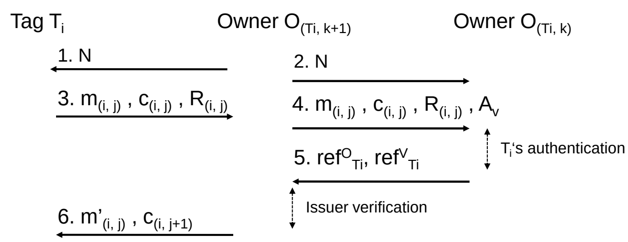

Figure 3: Ownership transfer in ROTIV

The ownership transfer protocol between $O_{(T_i,k)}$ and $O_{(T_i,k+1)}$ for tag T_i proceeds as follows:

1) The new owner $O_{(T_i,k+1)}$ sends a nonce N to tag T_i.

2) T_i replies with $c_{(i,j)} = (u_{(i,j)}, v_{(i,j)})$, a hash $m_{(i,j)}$ and a random number $R_{(i,j)}$.

3) $O_{(T_i,k+1)}$ selects a random number r_v and computes $A_v = u_{(i,j)}^{r_v}$. $O_{(T_i,k+1)}$ sends N, $R_{(i,j)}$, $c_{(i,j)}$, $m_{(i,j)}$ and A_v to T_i's owner O_{k}.

4) Provided with N, $R_{(i,j)}$, $c_{(i,j)}$ and $m_{(i,j)}$, $O_{(T_i,k)}$ authenticates T_i. If the authentication fails, $O_{(T_i,k)}$ informs $O_{(T_i,k+1)}$ who re-sends his first message to T_i. Otherwise, $O_{(T_i,k)}$ provides $O_{(T_i,k+1)}$ with the following:

$$\text{ref}_{T_i}^O = (k_i^{\text{old}}, k_i^{\text{new}}, x_i, y_i) = (k_{(i,j)}, k_{(i,j)}, t_i, h_i^x = H(t_i)^x)$$
$$\text{ref}_{T_i}^V = (A_{(i,j)}, B_{(i,j)}, C_{(i,j)}) = (t_i, h_i^x, A_v^{\alpha_k})$$

5) $O_{(T_i,k+1)}$ checks if the data provided by $O_{(T_i,k)}$ is valid by verifying whether the following equations hold:

$$e(H(x_i), g_2^x) = e(y_i, g_2) \quad (9)$$

If not $O_{(T_i,k+1)}$ aborts the ownership transfer protocol.

Otherwise, Provided with $\text{ref}_{T_i}^V$, $O_{(T_i,k+1)}$ verifies whether the issuer of T_i is $\mathcal{I}$.

If the verification fails, $O_{(T_i,k+1)}$ aborts the ownership transfer. If not, $O_{(T_i,k+1)}$ finishes the authentication with T_i.

7) To finish the authentication with T_i, $O_{(T_i,k+1)}$ chooses a new random number $r_{(i,j+1)} \in \mathbb{F}_q^*$ and computes:

$$c_{(i,j+1)} = (u_{(i,j+1)}, v_{(i,j+1)}) = (g_1^{r_{(i,j+1)}}, y_i g_1^{\alpha_{k+1}^2 r_{(i,j+1)}})$$
$$m'_{(i,j)} = \mathrm{HMAC}_{k_{(i,j)}}(R_{(i,j)}, c_{(i,j+1)})$$

$c_{(i,j+1)}$ is the encryption of y_i with $O_{(T_i,k+1)}$ public key: $g_1^{\alpha_{k+1}^2}$.

$O_{(T_i,k+1)}$ sends $(c_{(i,j+1)}, m'_{(i,j)})$ to T_i and updates its database $\mathcal{D}_{k+1}$ as in the authentication protocol presented above. Upon receiving $c_{(i,j+1)}$ and $m'_{(i,j)}$, T_i authenticates $O_{(T_i,k+1)}$. If the authentication succeeds T_i updates its state accordingly.

Note. In order to prevent the old owner $O_{(T_i,k)}$ from tracing the tag later in the future, the new owner $O_{(T_i,k+1)}$ has to run a mutual authentication with T_i *outside* the range of $O_{(T_i,k)}$ after the ownership transfer. In this manner, T_i and $O_{(T_i,k+1)}$ will share a symmetric key that $O_{(T_i,k)}$ cannot retrieve without physical access to T_i.

4. PRIVACY ANALYSIS

4.1 Forward Unlinkability

THEOREM 1 (FORWARD UNLINKABILITY). *ROTIV provides forward unlinkability under the SXDH assumption (DDH is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$).*

PROOF. Assume that there is an adversary $\mathcal{A}(r, s, \epsilon)$ who succeeds in the forward unlinkability experiment with a non negligible advantage ϵ. We will now construct an adversary $\mathcal{A}'(\frac{\epsilon}{2})$, using $\mathcal{A}$ as a subroutine, who breaks the DDH assumption in $\mathbb{G}_1$, therewith contradicting the SXDH assumption.

Let $\mathcal{O}_{\mathrm{DDH}}$ be an oracle that selects elements $\alpha, \beta \in \mathbb{F}_q$. Furthermore, $\mathcal{O}_{\mathrm{DDH}}$ sets $\gamma = \alpha\beta$ in 50% of the queries or selects a random $\gamma \in \mathbb{F}_q$ in the remaining 50% of the queries. $\mathcal{O}_{\mathrm{DDH}}$ returns the tuple $(g_1, g_1^\alpha, g_1^\beta, g_1^\gamma)$. Adversary $\mathcal{A}'$ breaks DDH, if given $(g_1, g_1^\alpha, g_1^\beta, g_1^\gamma)$, $\mathcal{A}'$ can tell whether $g_1^\gamma = g_1^{\alpha\beta}$.

Rationale: The idea of the proof is to build a ROTIV system with an issuer $\mathcal{I}$ of secret key g_2^x, and an owner O whose public key is g_1^α. A tag T_i in ROTIV therefore stores a ciphertext $c_{(i,j)} = (g_1^{r_{(i,j)}}, h_i^x g_1^{\alpha^{r_{(i,j)}}})$. To break DDH, $\mathcal{A}'$ stores in T_b in the challenge phase, a ciphertext $c_{(b,j+1)} = (g_1^\beta, h_b g_1^\gamma)$.

If $\gamma = \alpha\beta$ and $\mathcal{A}$'s advantage ϵ in breaking ROTIV is non-negligible, $\mathcal{A}$ will be able to output a correct guess for b. Therefore, $\mathcal{A}'$ will be able to break DDH.

Construction: First, $\mathcal{A}'$ queries $\mathcal{O}_{\mathrm{DDH}}$ to receive $(g_1, g_1^\alpha, g_1^\beta, g_1^\gamma)$.

Now, $\mathcal{A}'$ simulates a complete ROTIV system for $\mathcal{A}$, i.e., issuer $\mathcal{I}$, owners, and tags. However for simplicity, we assume here that all tags in the simulation belong to the same owner O. $\mathcal{A}'$ issues tags. He randomly selects $x \in \mathbb{F}_q$. Here, x represents the secret key of the issuer.

To issue a tag T_i in the simulation, $\mathcal{A}'$ randomly selects $t_i, r_{i,0}$ and $k_{(i,0)} \in \mathbb{F}_q$, computes $h_i = H(t_i)$, and $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)})$ $= (g_1^{r_{(i,0)}}, h_i^x (g_1^\alpha)^{r_{(i,0)}}) = (g_1^{r_{(i,0)}}, h_i^x g_1^{\alpha r_{(i,0)}})$. Finally, $\mathcal{A}'$ stores $s_{(i,0)} = (k_{(i,0)}, c_{(i,0)})$ in tag T_i.

Therefore, T_i is a tag issued by an issuer with public key g_2^x and owned by owner O with a public key $pk = (g_1^\alpha, g_2^{\sqrt{\alpha}})$. $\mathcal{A}'$ publishes g_1^α, g_2^r as public key for O, where r is selected randomly in $\mathbb{F}_q$. Note that given the DDH assumption in $\mathbb{G}_2$, $\mathcal{A}$ cannot tell if g_2^r is equal to $g_2^{\sqrt{\alpha}}$.

Note that, at this time, $\mathcal{A}'$ cannot compute the secret key $sk = \sqrt{\alpha}$ of O. Still, $\mathcal{A}'$ can successfully simulate O: as $\mathcal{A}'$ knows the symmetric keys k shared with tags, $\mathcal{A}'$ can compute the HMAC and authenticate tags.

• In the learning phase of the forward unlinkability experiment, $\mathcal{A}'$ simulates $\mathcal{O}_\mathcal{T}$ and provides $\mathcal{A}$ with tags T_0 and T_1. $\mathcal{A}$ can eavesdrop on T_0 and T_1 a total of s times. $\mathcal{A}'$ provides $\mathcal{A}$ with r tags T'_i. $\mathcal{A}$ can read T'_i's internal state, modify it and eavesdrop on T'_i's interactions with its owner O (simulator $\mathcal{A}'$) up to s times.

• In the challenge phase, $\mathcal{A}'$ starts authentications outside the range of $\mathcal{A}$ with T_0 by sending a nonce N_0 and with T_1 by sending a nonce N_1. We assume T_0 stores $s_{(0,j)} = (k_{(0,j)}, c_{(0,j)})$ and T_1 stores $s_{(1,j)} = (k_{(1,j)}, c_{(1,j)})$.

• At the end of an authentication, $\mathcal{A}'$ updates the state of T_0 and T_1 as follows: $s_{(i,j+1)} = (k_{(i,j+1)}, c_{(i,j+1)})$, $i \in \{0, 1\}$, where $k_{(i,j+1)} = G(N_i, k_{(i,j)})$ and $c_{(i,j+1)} = (g_1^\beta, h_i^x g_1^\gamma)$.

• $\mathcal{A}'$ simulates $\mathcal{O}_{\mathrm{flip}}$ and gives T_b to $\mathcal{A}$. $\mathcal{A}$ can read the internal state of T_b, modify T_b's state and eavesdrop T_b's interactions up to s times.

• $\mathcal{A}'$ simulates $\mathcal{O}_\mathcal{T}$ and provides $\mathcal{A}$ with r tags T''_i. Again, $\mathcal{A}$ can read the internal state of T''_i, modify T''_i's state and eavesdrop on T''_i's authentications up to s times.

• Given that $\mathcal{A}$ does not have access to $N_i, i \in \{0, 1\}$, $k_{(i,j+1)} = k_{(i,j+1)} = G(k_{(i,j)}, N_i)$ cannot give $\mathcal{A}$ any information about T_b's past interactions. So, $\mathcal{A}$ must focus on ciphertext $c_{(i,j+1)}$.

• At the end of the challenge phase, $\mathcal{A}$ outputs his guess of b.

If $\gamma = \alpha\beta$, the state $c_{(b,j+1)} = (g_1^\beta, h_b^x g_1^{\alpha\beta})$ corresponds to a valid state of tag T_b. Therefore, $\mathcal{A}$ can output a correct guess for the tag corresponding to T_b with non negligible advantage ϵ.

If $\gamma \neq \alpha\beta$, the probability that $\mathcal{A}'$ can break the DDH is a random guess, i.e., $\frac{1}{2}$.

In general, given two events $\{E_1, E_2\}$, the probability that event E_1 occurs is $Pr(E_1) = Pr(E_1|E_2) \cdot Pr(E_2) + Pr(E_1|\overline{E_2}) \cdot Pr(\overline{E_2})$.

Let E_1 be the event that $\mathcal{A}'$ can break DDH, and E_2 is the event that $\gamma = \alpha\beta$ holds. The probability of event E_2 is $\frac{1}{2}$.

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

Therefore, with $\mathcal{A}$'s non negligible advantage in breaking forward unlinkability of ROTIV, $\mathcal{A}'$'s advantage in breaking DDH in $\mathbb{G}_1$ is also non negligible.

4.2 Backward Unlinkability

THEOREM 2 (BACKWARD UNLINKABILITY). *ROTIV provides backward unlinkability under the SXDH assumption.*

PROOF SKETCH. The idea behind this proof is similar to the proof above. An adversary $\mathcal{A}'$ can break DDH in $\mathbb{G}_1$, using an adversary $\mathcal{A}$ who breaks ROTIV.

Unlike the forward unlinkability experiment $\mathcal{A}$ can read the internal state of T_0 and T_1 in the learning phase of the backward unlinkability experiment. That is, $\mathcal{A}$ knows T_0's secret key and T_1's secret key.

When T_0 and T_1 are authenticated outside the range of $\mathcal{A}$, their symmetric keys are updated using the nonces sent by $\mathcal{A}'$. Therefore, even if $\mathcal{A}$ knows the keys stored in T_0 and T_1 before the challenge phase, he cannot use this information to distinguish keys in the challenge phase. Therefore, the backward unlinkability of ROTIV boils down to solving a DDH instance in $\mathbb{G}_1$.

5. SECURITY ANALYSIS

5.1 Ownership Transfer

5.1.1 Secure authentication

THEOREM 3 (SECURE AUTHENTICATION). *The ownership transfer protocol in ROTIV provides secure authentication under the security of HMAC.*

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

HMAC Security: a secure HMAC fulfills the two following properties:

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

2.) Indistinguishability: Let $\mathcal{O}^{\text{distinguish}}_{\text{HMAC}_k}$ be an oracle, when provided with a message m, it flips a coin $b \in \{0, 1\}$ and returns a message m' such that: if b = 0, it returns a random number. If b = 1, it returns $\text{HMAC}_k(m)$. $\mathcal{A}'$ cannot tell if m' is a random number or $m' = \text{HMAC}_k(m)$ without having the secret key k.

PROOF. To simplify the proof, we assume that the key k_i shared between tag T_i and T_i's owner is not updated after each authentication. As the key update is only required to achieve privacy and exclusive ownership, it is irrelevant for the authentication proof.

We show that if $\mathcal{A}(r, \epsilon)$ is able to break the security of the authentication scheme with non-negligible advantage, then we can construct adversary $\mathcal{A}'(r', \epsilon)$ that breaks the resistance to existential forgery of HMAC with non-negligible advantage ϵ.

Rationale: To break the HMAC existential forgery, $\mathcal{A}'$ simulates both the challenge tag T_c, and T_c's owner. To compute the HMAC, $\mathcal{A}'$ queries the oracle $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$. If $\mathcal{A}$'s advantage ϵ is non negligible in succeeding in the authentication experiment, $\mathcal{A}'$ will be able to compute a valid HMAC HMAC for a message m which he has not seen before.

Therefore, to break HMAC security $\mathcal{A}'$ answers with the pair $(m, \text{HMAC}(m))$.

Construction: $\mathcal{A}'$ simulates issuer $\mathcal{I}$ and creates tags:

1) $\mathcal{A}'$ selects randomly $x \in \mathbb{F}_q$. Here, x will be the secret key of the issuer.

2) $\mathcal{A}'$ selects randomly $t_i \in \mathbb{F}_q$ and computes $h_i = H(t_i)$. Finally, $\mathcal{A}'$ selects randomly $\alpha_k \in \mathbb{F}_q$ and computes : $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)}) = (g_1^{r(i,0)}, h_i^x g_1^{\alpha_k^2 r(i,0)})$.

Finally, $\mathcal{A}'$ selects randomly $k_i \in \mathbb{F}_q$ and stores $s_{(i,0)} = (k_i, c_{(i,0)})$ into T_i.

• $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and provides $\mathcal{A}$ with tag T_c.

• $\mathcal{A}'$ will simulate both T_c and $O_{(T_c,k)}$ in the rest of experiment.

• In the learning phase, $\mathcal{A}'$ starts mutual authentications with T_c that $\mathcal{A}$ can eavesdrop on or alter by injecting fake messages. $\mathcal{A}$ can start authentications with T_c while impersonating T_c's owner. He can as well start authentications with $O_{(T_c,k)}$ while impersonating T_c.

• When T_c receives a nonce N, $\mathcal{A}'$ generates a random number R and queries the oracle, $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with $m = (N, R, c_{(i,j)})$, and $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\sigma = \text{HMAC}_k(m)$. $\mathcal{A}'$ replies with $R, c_{(i,j)}$ and σ.

• When $O_{(T_c,k)}$ receives $(R, c_{(i,j)}, \sigma)$, $\mathcal{A}'$ identifies T_c by decrypting $c_{(i,j)}$, if the identification fails $\mathcal{A}'$ aborts the authentication. Otherwise, $\mathcal{A}'$ queries the oracle with message $m = (N, R, c_{(i,j)})$, and $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\text{HMAC}_k(m)$, $\mathcal{A}'$ checks whether $\sigma = \text{HMAC}_k(m)$, if not, $\mathcal{A}'$ aborts authentication. Otherwise, $\mathcal{A}'$ computes $c_{(i,j+1)}$ and queries $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with message $m' = (R, c_{(i,j+1)})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\sigma' = \text{HMAC}_k(m')$. $\mathcal{A}'$ sends the last message of authentication $(R, c_{(i,j+1)}, \sigma')$

• When T_c receives the last message $(R, c_{(i,j+1)}, \sigma')$, $\mathcal{A}'$ queries $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with $m' = (R, c_{(i,j+1)})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\text{HMAC}_k(m')$. $\mathcal{A}'$ checks whether $\sigma' = \text{HMAC}_k(m)$. If not, $\mathcal{A}'$ aborts the authentication. Otherwise, he writes $c_{(i,j+1)}$ into T_c.

• In the challenge phase, $\mathcal{A}$ runs a mutual authentication, either with

1) T_c while impersonating $O_{(T_c,k)}$. $\mathcal{A}$ sends a nonce N' to T_c. $\mathcal{A}'$, generates R', and queries the oracle $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with message $m = (N', R', c_{(i,j)})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\sigma = \text{HMAC}_k(m)$. $\mathcal{A}'$ sends R', $c_{(i,j)}$ and σ to $\mathcal{A}$.

$\mathcal{A}$ replies with (c', σ').

If $\mathcal{A}$'s advantage ϵ in impersonating T_c's owner is not negligible, we will have $\sigma' = \text{HMAC}_k(R, c')$. Therefore, to break the existential forgery of HMAC_k with non negligible advantage ϵ, $\mathcal{A}'$ simply outputs $((R, c'), \sigma')$. This leads to a contradiction under the security of HMAC.

2) or with T_c's owner while impersonating T_c. $\mathcal{A}'$ sends a fresh nonce N to $\mathcal{A}$. Upon receiving N, $\mathcal{A}$ generates a random number R. $\mathcal{A}$ sends with R, c' and σ to $\mathcal{A}'$.

If $\mathcal{A}$'s advantage ϵ in impersonating T_c is non negligible, we have $\sigma = \text{HMAC}_k(N, R, c')$. Therefore, to break the existential forgery of HMAC_k, $\mathcal{A}'$ can output $((N, R, c'), \sigma)$ and is successful with non negligible advantage ϵ. This also leads to a contradiction of the HMAC security assumption.

5.1.2 Exclusive Ownership

THEOREM 4 (EXCLUSIVE OWNERSHIP). *The ownership transfer protocol in ROTIV provides exclusive ownership under the security of the hash function H.*

PROOF. Assume there is an adversary $\mathcal{A}(r, s, \epsilon)$ who succeeds in the exclusive ownership game with a non negligible advantage ϵ. If so, we can construct an adversary $\mathcal{A}'$ who breaks the "one wayness" of H with a non negligible advantage ϵ.

One Wayness: Let $\mathcal{O}_H$ be an oracle that, when queried, returns a hash $H(m)$. $\mathcal{A}'$ breaks the one wayness of H, if given $H(m)$, he outputs m with non negligible advantage over simple guessing.

Rationale: To break the one wayness of H, $\mathcal{A}'$ queries the oracle $\mathcal{O}_H$ which returns a hash h_n. $\mathcal{A}'$ creates a tag T_c such that $s_{(0,n)} = (k_{(0,n)}, c_{(0,n)})$, where $c_{(0,n)} = (g_1^{r(0,n)}, h_n^x g_1^{\alpha_k^2 r(0,n)})$. If $\mathcal{A}$ has a non negligible advantage ϵ in succeeding in the exclusive ownership transfer, $\mathcal{A}$ will be able to transfer the ownership of T_c with a non negligible advantage. That is, $\mathcal{A}$ outputs valid ownership references for T_c, $\text{ref}^{O}_{T_c} = (t_n, h_n^x, k_{\text{old}}, k_{\text{new}})$, where $h_n = H(t_n)$.

To break H's one wayness, $\mathcal{A}'$ outputs t_n.

Construction: $\mathcal{A}'$ simulates the issuer $\mathcal{I}$ and creates $(n-1)$ tags.

1) $\mathcal{A}'$ selects randomly $x \in \mathbb{F}_q$. x will be the secret key of the issuer.

2) For each tag T_i, $\mathcal{A}'$ selects randomly $t_i \in \mathbb{F}_q$ and computes $h_i = H(t_i)$. $\mathcal{A}'$ selects randomly $\alpha_k \in \mathbb{F}_q$ and computes $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)}) = (g_1^{r(i,0)}, h_i^x g_1^{\alpha_k^2 r(i,0)})$. Also, $\mathcal{A}'$ selects randomly $k_{(i,0)} \in \mathbb{F}_q$ and stores $s_{(i,0)} = (k_{(i,0)}, c_{(i,0)})$ into T_i. $\mathcal{A}'$ creates

$\mathrm{ref}^{O}_{T_i} = (t_i, h_i^x,$
$k_{(i,0)}, k_{(i,0)})$.

3) $\mathcal{A}'$ queries $\mathcal{O}_H$ that returns hash h_n. $\mathcal{A}'$ selects a random number $r_{(n,0)}$ and computes $c_{(n,0)} = (u_{(n,0)}, v_{(n,0)}) = (g_1^{r_{(n,0)}}, h_n^x g_1^{\alpha_k^2 r_{(n,0)}})$.

Therewith, $\mathcal{A}'$ selects randomly $k_{(n,0)} \in \mathbb{F}_q$ and stores $s_{(n,0)} = (k_{(n,0)}, c_{(n,0)})$ into tag T'_c.

• $\mathcal{A}$ enters the learning phase. $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$. $\mathcal{A}'$ selects randomly a tag T_i from the n tags he created and checks if $T_i = T'_c$. If so, $\mathcal{A}'$ aborts the experiment, otherwise, $\mathcal{A}'$ provides $\mathcal{A}$ with tag T_i.

• Simulating $\mathcal{O}_{\mathrm{own}}$, $\mathcal{A}'$ provides $\mathcal{A}$ with T_i's ownership references. $\mathcal{A}$ can read T_i's state, modify it, and run mutual authentications with T_i.

• In the challenge phase, $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and selects randomly a tag T_c for which $\mathcal{A}$ did not query the oracle $\mathcal{O}_{\mathrm{own}}$.

If $T_c \neq T'_c$, $\mathcal{A}'$ stops the experiment. Otherwise, $\mathcal{A}'$ provides $\mathcal{A}$ with T_c.

• $\mathcal{A}$ now can read T_c's internal state for up to s times, he can as well eavesdrop on T_c.

• $\mathcal{A}'$ simulates $\mathcal{O}_{\mathbb{O}}$ and returns an owner O_c.

• At the end of the challenge phase, $\mathcal{A}$ runs an ownership transfer with O_c.

If $\mathcal{A}$'s advantage in breaking the exclusive ownership is non negligible, $\mathcal{A}$ will provide O_c during the ownership transfer protocol with $\mathrm{ref}^{O}_{T_c} = (t_n, h_n^x, k_{\mathrm{old}}, k_{\mathrm{new}})$, where $h_n = H(t_n)$.

Therefore, to break the one wayness of H, $\mathcal{A}'$ outputs t_n.

Note that $\mathcal{A}'$ succeeds in breaking H, if he does not stop the experiment. The probability that $\mathcal{A}'$ does not stop the experiment corresponds to not choosing T'_c in the learning phase, and choosing T'_c in the challenge phase. The probability that $\mathcal{A}'$ does not choose T'_c in the learning phase is $(1-\frac{1}{n})^r$. The probability that $\mathcal{A}'$ chooses T'_c in the challenge phase is $\frac{1}{n}$

Hence, $\mathcal{A}'$'s advantage is

$$\epsilon' = \Pr(\mathcal{A}' \text{ does not abort the experiment})\epsilon = (1-\frac{1}{n})^r \frac{1}{n}\epsilon$$

This leads to a contradiction under the security of H.

Note. Someone could argue that if n, i.e., the total number of tags, is large, the advantage ϵ will not be negligible, even if ϵ' is negligible.

In the following, we show that if ϵ' is negligible, so is ϵ.

Let k be the security parameter of H. As ϵ' is negligible we have:

$\forall\, c$, $\exists\, N_c$ such that $\forall\, k > N_c$: $|\epsilon'| \leq \frac{1}{k^c}$.

Therefore,

$\forall\, c$, $\exists\, N_c, \forall\, k > N_c$: $|\epsilon' \frac{n}{(1-\frac{1}{n})^r}| \leq \frac{n}{k^c(1-\frac{1}{n})^r}$.

$\forall\, c$, $\exists\, N_c, \forall\, k > N_c$: $|\epsilon| \leq \frac{n}{k^c(1-\frac{1}{n})^r}$.

Note that for $\forall\, k > 2n^{\frac{1}{r}} > \frac{n^{\frac{1}{r}}}{1-\frac{1}{n}} \Rightarrow \frac{n}{(1-\frac{1}{n})^r} < k^r$, hence, $\forall\, c$, $\exists\, N_c, \forall\, k > N_c$ and $k > 2n^{\frac{1}{r}}$: $|\epsilon| \leq \frac{k^r}{k^c} \leq \frac{1}{k^{c-r}}$.

That is, $\forall\, c$, $\exists\, N_c, \forall\, k > \max(N_c, 2n^{\frac{1}{r}})$: $|\epsilon| \leq \frac{1}{k^{c-r}}$.

If we denote $c' = c - r$, and $N_{c'} = \max(N_c, 2n^{\frac{1}{r}})$, we have, $\forall\, c'$, $\exists\, N_{c'}, \forall\, k > N_{c'}$: $|\epsilon| \leq \frac{1}{k^{c'}}$.

Consequently, ϵ is negligible.

5.2 Issuer verification protocol

THEOREM 5 (ISSUER VERIFICATION SECURITY). *The issuer verification protocol in ROTIV is secure under the BCDH assumption.*

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

Square BCDH assumption: We use a modified version of the BCDH assumption: square BCDH, sq-BCDH for short. The sq-BCDH assumption states that given g_1, g_2, g_1^x, g_2^x, g_1^y, one cannot compute $e(g_1, g_2)^{x^2 y}$.

We show in the appendix the equivalence between BCDH and sq-BCDH.

Rationale: If $\mathcal{A}$ has a non negligible advantage in succeeding in the issuer verification experiment, $\mathcal{A}$ will be able to output valid verification references for a fake tag T_c which he creates. That is, $\mathrm{ref}^{V}_{T_c} = (A_c, B_c, C_c) = (t_c, h_c^x, C_c)$, where h_c is the hash of t_c. Therefore, to break the sq-BCDH assumption, $\mathcal{A}'$ simulates the outputs of H, during the issuer verification experiment. When $\mathcal{A}$ queries H with T_c's identifier t_c, $\mathcal{A}'$ selects randomly $r_c \in \mathbb{F}_q$ and outputs $h_c = H(t_c) = g_1^{yr_c}$.

At the end of the challenge phase, $\mathcal{A}$ outputs a valid tuple: $\mathrm{ref}^{V}_{T_c} = (A_c, B_c, C_c) = (t_c, h_c^x, C_c) = (t_c, g_1^{xyr_c}, C_c)$. To break sq-BCDH $\mathcal{A}'$ outputs $e(g_1, g_2)^{x^2 y} = e(g_1^{xyr_c}, g_2^x)^{r_c^{-1}}$.

Random oracle H**:** On a query $H(t)$, if t has never been queried before, $\mathcal{A}'$ picks $r_t \in \mathbb{F}_q$ and stores the pair (t, r_t) in a table T_H. Then, $\mathcal{A}'$ flips a random coin $\mathrm{coin}(t) \in \{0,1\}$ such that: $\mathrm{coin}(t) = 1$ with probability p, and is equals to 0 with probability $1-p$. To compute $H(t)$, $\mathcal{A}'$ checks $\mathrm{coin}(t)$: if $\mathrm{coin}(t) = 0$, $\mathcal{A}'$ looks up r_t in T_H, and answers $H(t) = g_1^{r_t}$. Otherwise, if $\mathrm{coin}(t) = 1$, $\mathcal{A}'$ answers with $H(t) = (g_1^y)^{r_t}$.

This motivates why in our protocol design, we use different hash functions for tags and issuer. That is, G for tags and H for issuer. If tags T_i implements the same hash function as the issuer, $\mathcal{A}$ will not need to query $\mathcal{A}'$ to get the output of H, he can only use one of the tags he controls to compute H.

Construction: $\mathcal{A}'$ first queries $\mathcal{O}_{\mathrm{sq-BCDH}}$ to receive $(g_1, g_2, g_1^x, g_2^x, g_1^y)$.

$\mathcal{A}'$ simulates an issuer $\mathcal{I}$ of public key g_2^x to create r tags T_i:

1) He selects randomly $t_i \in \mathbb{F}_q$, then computes $h_i = H(t_i)$ as above. If $\mathrm{coin}(t_i) = 1$ $\mathcal{A}'$ aborts the experiment. Otherwise, $\mathcal{A}'$ computes h_i^x. To do so, he looks up his table T_H for t_i, gets r_{t_i}, and computes $h_i^x = (g_1^x)^{r_{t_i}}$. $\mathcal{A}'$ selects randomly α_k , $r_{(i,0)} \in \mathbb{F}_q$ and computes $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)}) = (g_1^{r_{(i,0)}}, h_i^x g_1^{\alpha_k^2 r_{(i,0)}})$.

Finally, $\mathcal{A}'$ chooses randomly a key $k_{(i,0)} \in \mathbb{F}_q$ and stores $s_{(i,0)} = (k_{(i,0)}, c_{(i,0)})$ into T_i.

2) $\mathcal{A}'$ stores the ownership references of tag T_i, $\mathrm{ref}^{O}_{T_i} = (k_{(i,0)}, k_{(i,0)}, h_i, h_i^x)$.

• $\mathcal{A}$ enters the learning phase.

• $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and provides $\mathcal{A}$ with r tags T_i.

• $\mathcal{A}'$ simulates $\mathcal{O}_{\mathrm{own}}$ and provides $\mathcal{A}$ with T_i's ownership references $\mathrm{ref}^{O}_{T_i}$. Provided with the ownership references $\mathcal{A}$ has full control of T_i, and he can now run authentications with T_i, issuer verification and ownership transfer for T_i.

• In the challenge phase, $\mathcal{A}'$ simulates the verifier $\mathcal{V}$.

• $\mathcal{A}$ is required to create a new tag T_c. Therefore, $\mathcal{A}$ selects randomly $t_c \in \mathbb{F}_q$ and queries H. To answer this query, $\mathcal{A}'$ flips a coin $\mathrm{coin}(t_c)$, if $\mathrm{coin}(t_c) = 0$, $\mathcal{A}'$ stops the experiment. Otherwise, $\mathcal{A}'$ selects randomly $r_c \in \mathbb{F}_q$ and answers with $h_c = (g_1^y)^{r_c} = g_1^{yr_c}$.

If $\mathcal{A}$'s advantage in breaking ROTIV verification protocol is non negligible, $\mathcal{A}$ will provide a valid verification reference $\mathrm{ref}^{V}_{T_c}$ for T_c during the issuer verification protocol. That is:

$$\mathrm{ref}^{V}_{T_c} = (A_c, B_c, C_c) = (t_c, h_c^x, C_c) = (t_c, g_1^{yxr_c}, C_c)$$

Finally, to break sq-BCDH, $\mathcal{A}'$ computes the following:

$$e(B_c, g_2^x)^{r_c^{-1}} = e(h_c^x, g_2^x)^{r_c^{-1}} = e(g_1^{yr_c{}^x}, g_2^x)^{r_c^{-1}} = e(g_1, g_2)^{x^2y}$$

Note that $\mathcal{A}'$ succeeds in breaking BCDH is he does not stop this experiment. $\mathcal{A}'$ does not stop the experiment, if for all the r tags T_i he created for the learning phase, $\mathrm{coin}(t_i) = 0$, and if for tag T_c $\mathrm{coin}(t_c) = 1$.

Therefore the probability that $\mathcal{A}'$ does not stop the experiment is $p(1-p)^r$. Thus, $\mathcal{A}'$'s advantage is:

$$\epsilon' = p(1-p)^r\epsilon$$

If ϵ is non negligible, so is ϵ'. This leads to a contradiction under the BCDH assumption.

6. RELATED WORK

Molnar et al. [13] address the problem of ownership transfer in RFID systems by relying on a trusted party during the actual transfer itself. When a reader reads a tag T, it sends the tag pseudonym he receives from T to the trusted party in order to identify T. If the reader is actually T's owner, the trusted party replies with T's identity. To transfer ownership of T, the owner of T and the new owner of T ask the trusted party. Once the ownership transfer of T takes place, the trusted center refuses identity requests from T's previous owner. The requirement for a trusted third party is, however, a drawback: in many scenarios, the availability of a trusted third party during tag ownership transfer is probably unrealistic.

Similar, Saito et al. [16] suggest two approaches for ownership transfer. To allow for ownership transfer for tag T, T's previous owner O provides the new owner O' with a symmetric key, then O' sends this key to a trusted third party. The trusted third party helps in finalizing the ownership transfer. Again, this approach, like Molnar et al. [13], relies on a trusted third party – a major drawback. Moreover, Saito et al. [16] require tamper resistant memory to store keys, Otherwise, anyone can impersonate the trusted third party.

The ownership transfer protocol introduced by Song [17] has been shown to have security weaknesses as highlighted by Peris-Lopez et al. [15].

Lim and Kwon [12] rely on hash chains to provide forward unlinkability during tag ownership. However, this scheme, as well as all the others mentioned above, does neither allow owners to verify, whether a tag T was issued by a legitimate party nor constant-time authentication. These are major contributions of the paper at hand.

7. CONCLUSION

In this paper, we presented ROTIV to address security and privacy issues related to RFID ownership transfer in supply chains. Moreover, ROTIV enables ownership transfer together with issuer verification. Such verification will prevent partners in a supply chain from injecting fake products. ROTIV's main idea is to store a signature of the issuer in tags that can be verified by every partner in the supply chain. Also, to allow for efficient ownership transfer, ROTIV comprises an efficient, constant time authentication protocol. To guarantee tag privacy, we use re-encryption and key update techniques. Despite the high security and privacy properties, ROTIV is lightweight and requires a tag to only evaluate a hash function.

References

[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 encryption. Cryptology ePrint Archive, Report 2005/417, 2005. `http://eprint.iacr.org/`.

[4] M. Burmester, B. de Medeiros, and R. Motta. Robust, anonymous RFID authentication with constant key-lookup. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 283–291, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-979-1.

[5] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, USA, 1985. Springer New York, Inc.

[6] S. Fouladgar and H. Afifi. An Efficient Delegation and Transfer of Ownership Protocol for RFID Tags. In *First International EURASIP Workshop on RFID Technology*, Vienna, Austria, September 2007.

[7] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156:3113–3121, September 2008. ISSN 0166-218X.

[8] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal re-encryption for mixnets. In *RSA Conference, Cryptographer's track*, pages 163–178. Springer, 2004.

[9] 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.

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

[11] Y. K. Lee, L. Batina, D. Singelée, and I. Verbauwhede. Low-Cost Untraceable Authentication Protocols for RFID. In Susanne Wetzel, Cristina Nita-Rotaru, and Frank Stajano, editors, *Proceedings of the 3rd ACM Conference on Wireless Network Security – WiSec'10*, pages 55–64, Hoboken, New Jersey, USA, March 2010. ACM, ACM Press.

[12] C. H. Lim and T. Kwon. Strong and Robust RFID Authentication Enabling Perfect Ownership Transfer. In *ICICS*, pages 1–20, 2006.

[13] D. Molnar, A. Soppera, and D. Wagner. A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 276–290. Springer Berlin / Heidelberg, 2006.

[14] R. Paise and S. Vaudenay. Mutual authentication in RFID: security and privacy. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 292–299, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-979-1.

[15] P. Peris-Lopez, J. C. Hernandez-Castro, J. M.E. Tapiador, T. Li, and Y. Li. Vulnerability analysis of rfid protocols for tag ownership transfer. *Computer Networks*, 54(9):1502 – 1508, 2010. ISSN 1389-1286.

[16] J. Saito, K. Imamoto, and K. Sakurai. Reassignment scheme of an rfid tag's key for owner transfer. In *Embedded and Ubiquitous Computing*, volume 3823 of *Lecture Notes in Computer Science*, pages 1303–1312. Springer Berlin / Heidelberg, 2005.

[17] B. Song. RFID Tag Ownership Transfer. In *Workshop on RFID Security – RFIDSec'08*, Budapest, Hungary, July 2008.

[18] S. Vaudenay. On privacy models for RFID. In *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, ASIACRYPT'07, pages 68–87, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-76899-8, 978-3-540-76899-9.

APPENDIX

THEOREM 6. *The BCDH assumption and the sq-BCDH assumption are equivalent.*

PROOF. Let $\mathcal{O}_{\mathrm{BCDH}}$ be an oracle, when queried selects randomly $a, a', b \in \mathbb{F}_q$, returns $g_1, g_1^a, g_1^{a'}, g_1^b, g_2, g_2^a, g_2^{a'}$.

An adversary $\mathcal{A}$ breaks BCDH, if given $(g_1, g_1^a, g_1^{a'}, g_1^b, g_2, g_2^a, g_2^{a'})$, he can compute $e(g_1, g_2)^{aa'b}$ with a non negligible advantage ϵ.

Let $\mathcal{O}_{\mathrm{sq-BCDH}}$ be an oracle, when queried selects randomly $a, b \in \mathbb{F}_q$ and returns $g_1, g_1^a, g_1^b, g_2, g_2^a$.

An adversary $\mathcal{A}$ breaks sq-BCDH, if given $(g_1, g_1^a, g_1^b, g_2, g_2^a)$, he can compute $e(g_1, g_2)^{a^2 b}$ with a non negligible advantage ϵ.

Note that sq-BCDH is an instance of BCDH, therefore, if there is an adversary $\mathcal{A}$ who breaks BCDH, the same adversary can break sq-BCDH.

In what follows, we show that if there is an adversary $\mathcal{A}$ who breaks sq-BCDH with a non negligible advantage ϵ, we build an adversary $\mathcal{A}'$ who breaks BCDH with advantage ϵ.

Construction: First, $\mathcal{A}'$ queries $\mathcal{O}_{\mathrm{BCDH}}$ to receive $(g_1, g_1^a, g_1^{a'}, g_1^{a'}, g_2, g_2^a, g_2^{a'})$.

• $\mathcal{A}'$ chooses randomly $t_1 \in \mathbb{F}_q$, and simulates $\mathcal{O}_{\mathrm{sq-BCDH}}$ by giving $\mathcal{A}$ $g_1, g_1^a, g_1^{t_1 b}, g_2, g_2^a$. $\mathcal{A}$ outputs $e(g_1, g_2)^{a^2 t_1 b}$, and $\mathcal{A}'$ computes $A_1 = (e(g_1, g_2)^{a^2 t_1 b})^{t_1^{-1}} = e(g_1, g_2)^{a^2 b}$.

• $\mathcal{A}'$ chooses randomly $t_2 \in \mathbb{F}_q$, simulates again $\mathcal{O}_{\mathrm{sq-BCDH}}$, and provides $\mathcal{A}$ with $g_1, g_1^{a'}, g_1^{t_2 b}, g_2$, $g_2^{a'}$. $\mathcal{A}$ outputs $A_2 = e(g_1, g_2)^{a'^2 t_2 b}$, and $\mathcal{A}'$ computes $A_2 = (e(g_1, g_2)^{a'^2 t_2 b})^{t_2^{-1}} = e(g_1, g_2)^{a'^2 b}$.

• $\mathcal{A}'$ selects randomly r_1 and r_2 in $\mathbb{F}_q$, and then, simulates $\mathcal{O}_{\mathrm{sq-BCDH}}$ and provides $\mathcal{A}$ with $g_1, g_1^{r_1 a + r_2 a'}, g_1^b, g_2, g_2^{r_1 a + r_2 a'}$. $\mathcal{A}$ outputs $A_3 = e(g_1, g_2)^{(r_1 a + r_2 a')^2 b} = e(g_1, g_2)^{(a r_1)^2 b + 2 r_1 r_2 a a' b + (a' r_2)^2 b}$.

• $\mathcal{A}'$ computes $B = A_1^{r_1^2} = e(g_1, g_2)^{(a r_1)^2 b}$, $C = A_2^{r_2^2} = e(g_1, g_2)^{(a' r_2)^2 b}$, and computes $D = \frac{A_3}{BC} = e(g_1, g_2)^{2 r_1 r_2 a a' b}$.

To solve BCDH, $\mathcal{A}$ outputs $D^{\frac{1}{r_1 r_2}} = e(g_1, g_2)^{aa'b}$.

Therefore, if $\mathcal{A}$ breaks sq-BCDH with a non negligible advantage ϵ, $\mathcal{A}'$ breaks BCDH with the same advantage ϵ.