ROTIV: RFID Ownership Transfer with Issuer Verification

Kaoutar Elkhiyaoui, Erik-Oliver Blass, and Refik Molva

Eurecom, Sophia-Antipolis, France
Email: {kaoutar.elkhiyaoui, erik-oliver.blass, refik.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. 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 secure ownership transfer against malicious owners. ROTIV offers as well 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 public key encryption to achieve constant time authentication and issuer verification. To assure privacy, ROTIV implements key update techniques and tag state re-encryption techniques, performed on the reader. ROTIV is especially designed for lightweight tags which 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 his site. To that effect, tags and partners in the supply chain must implement a secure ownership transfer protocol.

A secure ownership transfer protocol should fulfill two main *security* requirements: **1)** mutual authentication between the owner of a tag T (partner in the supply chain) and tag T to tell apart legitimate tags from counterfeits. **2)** exclusive ownership: non-authorized parties must not be able to transfer the ownership of tag T without the consent of T's owner. Furthermore, ownership transfer must be privacy preserving. It must ensure **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 tag ownership transfer as previously addressed in [13, 12, 6, 17], this paper proposes an *efficient* ownership transfer protocol that also allows a party possessing the right references 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.

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]. Previously proposed tag/reader authentication protocols that achieve constant time authentication rely on public key cryptography performed on the tag as in [11]. However, RFID tags are constrained devices that cannot implement asymmetric cryptography.

The above schemes are designed to be privacy preserving against a *strong* adversary as defined by Juels and Weis [8], who can *continuously* eavesdrop on tags' communications. We claim that such an adversary is unrealistic in distributed supply chains which is the targeted setting by ROTIV. In ROTIV, we relax some privacy requirements to achieve mutual authentication in constant time while the tag performs only *symmetric* cryptographic operations (hash functions).

In ROTIV, 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 current owner. In order to ensure tag privacy, we update T's state after each successful authentication. Moreover, each tag T in ROTIV is associated with a set of ownership references. T's ownership references allow T's owner to authenticate T and to transfer T's ownership. Finally, to allow tag issuer verification by third parties, a tag T stores an encryption of the issuer's signature. Provided with some trapdoor information from T's owner(the randomness used to encrypt $\mathcal{I}$'s signature), a third party verifier can verify whether the signature stored on T corresponds to a legitimate issuer or not.

In summary, ROTIV's contributions are:

- ownership transfer that ensures both tag forward unlinkability against the tag's new owner and tag backward unlinkability against the tag's previous owner.
- a *privacy-preserving*, and *constant time* authentication while tags are only required to compute a hash function.
- contrary to related work [16, 13, 6, 10], ROTIV does not require a trusted third party to perform tag ownership transfer.
- issuer verification protocol that allows prospective owners of a tag T to check the identity of the party issuing T.
- formal definitions of privacy and security requirements of tag ownership transfer.
- formal proofs of ROTIV security and privacy.

2 RFID ownership transfer with issuer verification

An ownership transfer protocol with issuer verification involves the following 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 set *ownership references* $\text{ref}^{O}_{T_i}$ that he gives to T_i's owner. $\mathcal{I}$ writes an initial state $s_{(i,0)}$ 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 $O_{(T_i,k)}$. 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 *verification references* $\text{ref}^{V}_{T_i}$ allowing $\mathcal{V}$ to verify the identity of the issuer of T_i.

2.2 RFID ownership transfer with issuer verification

Secure ownership transfer raises four major requirements as follows:

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, a prospective owner $O_{(T_i,k+1)}$, 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, $O_{(T_i,k+1)}$ is able to verify whether T_i has been originally issued by $\mathcal{I}$.

3 Problem statement

Recently proposed protocols on RFID tag ownership transfer [12, 6, 17] rely on symmetric primitives to perform privacy preserving mutual authentication and secure ownership transfer. As depicted in Figure 1, a tag T_i in these protocols

- stores a state $s_{(i,j)} = k_{(i,j)}$. This state corresponds to a secret key which is shared between T_i and T_i's owner $O_{(T_i,k)}$.

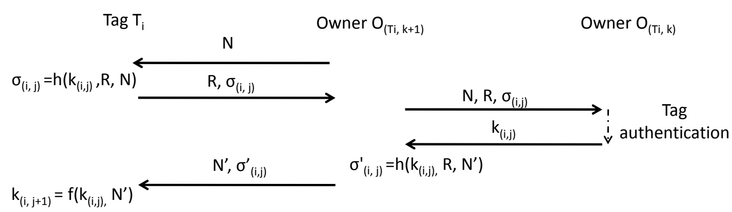

Fig. 1. Ownership transfer protocol

- computes a secure symmetric primitive h that is used to authenticate mutually T_i and $O_{(T_i,k)}$ using the secret key $k_{(i,j)}$.
- computes a function f that is used to update the secret key of T_i after a successful mutual authentication.

However, such protocols suffer from inherent limitations:

1) Linear complexity: As previously proposed protocols in [12, 17, 10] use symmetric primitives to authenticate a tag T_i, an owner has to try all the tags' keys in his database to authenticate T_i. Thus, in these schemes the authentication takes a linear time in the number of tags.

2) Denial of service: To ensure forward unlinkability, tag T_i updates its key $k_{(i,j)}$ using a secure hash function g even if the authentication with its owner $O_{(T_i,k)}$ is not successful as shown by Ohkubo et al. [14]. Also, $O_{(T_i,k)}$ keeps a limited set of η keys $(k_{(i,j+1)}, k_{(i,j+2)}, ..., k_{(i,j+\eta)}) = (g(k_{(i,j)}), g^2(k_{(i,j)}), ..., g^{\eta}(k_{(i,j)}))$ in his database $\mathcal{D}_k$ after each successful authentication with T_i. Thus, $O_{(T_i,k)}$ will still be able to authenticate T_i even if the authentication fails up to $\eta - 1$ times. However, an adversary can query T_i up to $p > \eta$ times, and therefore desynchronize T_i and $O_{(T_i,k)}$.

3) No tag issuer verification: Without tag issuer verification, owners and therewith partners in the supply chain will be able to inject tags that were not issued by trusted parties. We claim that in the real world, the prospective owner of tag T_i will require verifying the origin of T_i before accepting it.

To cope with these limitations we propose ROTIV. To achieve constant time authentication, a tag T_i in ROTIV stores in addition to its symmetric key $k_{(i,j)}$, an Elgamal ciphertext $c_{(i,j)}$ of T_i's identification information. When T_i is queried, it replies with $c_{(i,j)}$ and an HMAC computed using $k_{(i,j)}$. The owner decrypts $c_{(i,j)}$ and identifies T_i. Once T_i is identified, the owner authenticates T_i through HMAC. Furthermore, to prevent denial of service, a tag in ROTIV does not update its symmetric key unless the authentication is successful. Finally, to provide tag issuer verification, the ciphertext $c_{(i,j)}$ encrypts the signature of T_i's identifier by the issuer.

Note that protocols presented above [12, 6, 17] are designed to be forward privacy preserving against a *strong adversary* that continuously monitors tags [8, 18, 15]. However, in order to achieve both constant time authentication and denial of service resistance while the tag only computes hash functions, ROTIV must consider a more *realistic* adversary model. The adversary cannot continuously monitor a tag, i.e., there is at least *one* communication between the tag and its owner that is *unobserved* by the adversary.

Hence, ROTIV defines new privacy and security requirements that will be further discussed in Section 5. These requirements are along the same lines as recent research on RFID security such as [8, 18, 15].

Now, we present ROTIV in §4, followed by our privacy and security models in §5.

4 ROTIV

ROTIV takes place in subgroups of elliptic curves that support bilinear pairings.

4.1 Preliminaries

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*: $\forall\, x, y \in \mathbb{Z}_q$, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$;
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 of $\mathbb{G}_T$.

ROTIV's security and privacy rely on two assumptions.

Definition 1 (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 $g_1, g_1^x, g_1^y, g_1^z \in \mathbb{G}_1$ and $g_2, g_2^x, g_2^y \in \mathbb{G}_2$ for random $x, y, z \in \mathbb{F}_q$, the probability to compute $e(g_1, g_2)^{xyz}$ is negligible.*

Definition 2 (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 hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Thus, ROTIV uses bilinear groups where DDH is hard, see Ballard et al. [3], Ateniese et al. [1, 2]. Such groups can be chosen as specific subgroups of MNT curves. Also, results by Galbraith et al. [7] indicate the high efficiency of such pairings.

4.2 ROTIV description

1. 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 $c_{(i,j)}$ is 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 T_i's secret key $k_{(i,j)}$. Upon receipt of $c_{(i,j)}$, $O_{(T_i,k)}$ uses his Elgamal secret key to decrypt $c_{(i,j)}$. After 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 $k_{(i,j)}$ of tag T_i in his database and verifies the HMAC sent by T_i. Therefore, 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 perform ownership transfer of tag T_i, the current owner $O_{(T_i,k)}$ of T_i gives $O_{(T_i,k+1)}$ T_i's ownership references $\text{ref}^O_{T_i}$ 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 updates the ciphertext stored on T_i in every authentication he runs with T_i, using Elgamal re-encryption mechanisms. Moreoever, T_i updates its key $k_{(i,j)}$ after each successful authentication.

Finally, to achieve tag issuer verification, the ciphertext $c_{(i,j)}$ stored on T_i encrypts a signature of $\mathcal{I}$ on T_i's identifier. To perform issuer verification for tag T_i, a verifier $\mathcal{V}$ is provided with the ciphertext $c_{(i,j)}$ stored in T_i along with some trapdoor information called verification references $\text{ref}^V_{T_i}$. Then, given $c_{(i,j)}$ and $\text{ref}^V_{T_i}$, $\mathcal{V}$ is able to verify if $c_{(i,j)}$ is an encrypted signature by $\mathcal{I}$ of T_i's identifier.

2. Description A ROTIV system comprises l owners $O_{(T_i,k)}$ and n tags T_i. Each tag T_i can evaluate a cryptographic hash function G to compute an HMAC. The HMAC is used to authenticate T_i and T_i's owner, and to update the symmetric key after each successful authentication.

In the rest of this section we use the notation $\text{HMAC}_k(m, m') = \text{HMAC}_\text{k}(\text{m}||\text{m}')$, where $||$ denotes concatenation.

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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing. The issuer chooses $x \in \mathbb{Z}_q^*$ and computes g_2^x. $\mathcal{I}$'s secret key is $sk = x$ 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 $(g_1^{\alpha_k^2}, g_2^{\alpha_k})$. The system supplies 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 $H : \mathbb{F}_q \to \mathbb{G}_1$, $\mathcal{I}$ 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 $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)})$. $\mathcal{I}$ gives $O_{(T_i,k)}$ tag T_i and the corresponding ownership references $\text{ref}^O_{T_i} = (k_i^{\text{old}}, k_i^{\text{new}}, x_i, y_i) = (k_{(i,0)}, k_{(i,0)}, t_i, h_i^x)$.

Before accepting the tag, $O_{(T_i,k)}$ reads T_i and checks if the ownership references verify the equation: $e(H(x_i), g_2^x) = e(y_i, g_2)$. If so, this implies that T_i is actually issued by $\mathcal{I}$, that is $y_i = H(x_i)^x$.

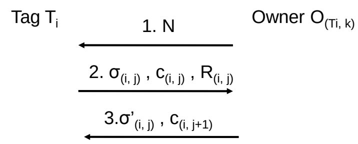

Fig. 2. Authentication in ROTIV

The owner $O_{(T_i,k)}$ adds an entry E_{T_i} for tag T_i in his database $\mathcal{D}_k$: $E_{T_i} = (y_i, \text{ref}^O_{T_i})$. 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 $g_1^{\alpha_k^2}$: $c_{(i,1)} = (u_{(i,1)}, v_{(i,1)}) = (g_1^{r_{(i,1)}}, y_i g_1^{\alpha_k^2 r_{(i,1)}})$. Therefore, $s_{(i,1)} = (k_{(i,1)} = k_{(i,0)}, c_{(i,1)})$.

Authentication protocol To authenticate a tag T_i, the owner $O_{(T_i,k)}$ decrypts the ciphertext $c_{(i,j)} = (u_{(i,j)}, v_{(i,j)})$ sent by T_i and gets y_i. Using y_i, $O_{(T_i,k)}$ identifies T_i and starts a hash-based mutual authentication. If the mutual authentication succeeds, both the owner $O_{(T_i,k)}$ and the tag T_i update their keys.

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 2.
Once T_i receives N, it generates a random number $R_{(i,j)} \in \mathbb{F}_q$. Using its secret key $k_{(i,j)}$, T_i computes: $\sigma_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(N, R_{(i,j)}, c_{(i,j)})$. This HMAC serves two purposes, it authenticates T_i and ensures the integrity of the message sent by T_i.

2. T_i replies with $(R_{(i,j)}, c_{(i,j)} = (u_{(i,j)}, v_{(i,j)}), \sigma_{(i,j)})$.
Upon receiving T_i's reply, the owner $O_{(T_i,k)}$ decrypts $c_{(i,j)}$ using his secret key α_k and gets $y_i = \frac{v_{(i,j)}}{(u_{(i,j)})^{\alpha_k^2}}$. $O_{(T_i,k)}$ 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 $\text{ref}^O_{T_i} = (k_i^{\text{old}}, k_i^{\text{new}}, t_i, h_i^x)$ in $\mathcal{D}_k$ and checks if: $\sigma_{(i,j)} = \text{HMAC}_{k_i^{\text{new}}}(N, R_{(i,j)}, c_{(i,j)})$ or $\sigma_{(i,j)} = \text{HMAC}_{k_i^{\text{old}}}(N, R_{(i,j)}, c_{(i,j)})$. If not, $O_{(T_i,k)}$ aborts authentication. If $\text{HMAC}_{k_i^{\text{old}}}(N, R_{(i,j)}, c_{(i,j)}) = \sigma_{(i,j)}$ 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:

$$c_{(i,j+1)} = (u_{(i,j+1)}, v_{(i,j+1)}) = (g_1^{r_{(i,j+1)}}, y_i g_1^{\alpha_k^2 r_{(i,j+1)}})$$
$$\sigma'_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(R_{(i,j)}, c_{(i,j+1)})$$

Finally, $O_{(T_i,k)}$ updates the symmetric keys k_i^{old} and k_i^{new} in his database $\mathcal{D}_k$: $(k_i^{\text{old}}, k_i^{\text{new}}) = (k_{(i,j)}, G(k_{(i,j)}, N))$.

3. $O_{(T_i,k)}$ sends $c_{(i,j+1)}$ and $\sigma'_{(i,j)}$ to T_i.
Once T_i receives $\sigma'_{(i,j)}$ and $c_{(i,j+1)}$, it checks if $\sigma'_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(R_{(i,j)}, c_{(i,j+1)})$. If not T_i aborts authentication. Otherwise, T_i updates its key such that $k_{(i,j+1)} = G(k_{(i,j)}, N)$ and rewrites its state $s_{(i,j+1)} = (k_{(i,j+1)}, c_{(i,j+1)})$.

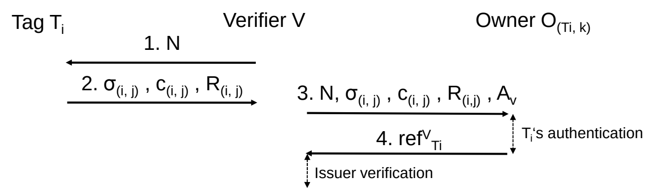

Fig. 3. Issuer verification in ROTIV

Desynchronization If the last message of the authentication protocol is lost, tag T_i will not update its state and therewith, T_i will not update its symmetric key $k_{(i,j)}$. However, as the owner $O_{(T_i,k)}$ keeps both keys $k_i^{\text{old}} = k_{(i,j)}$ and $k_i^{\text{new}} = G(k_{(i,j)}, N)$, $O_{(T_i,k)}$ can always re-synchronize with T_i using k_i^{old}.

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

1. $\mathcal{V}$ sends a nonce N to T_i, as depicted in Figure 3.
Upon receiving N, 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 random number $R_{(i,j)}$, and $\sigma_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(N, R_{(i,j)}, c_{(i,j)})$.

2. Once $\mathcal{V}$ receives T_i's reply, he chooses a random number $r_v \in \mathbb{F}_q^*$ and computes $A_v = (u_{(i,j)})^{r_v} = g_1^{r_{(i,j)} r_v}$.

3. Then, $\mathcal{V}$ sends $N, R_{(i,j)}, c_{(i,j)}, \sigma_{(i,j)}$ along with A_v to $O_{(T_i,k)}$.
When receiving the tuple $(N, R_{(i,j)}, c_{(i,j)}, \sigma_{(i,j)}, A_v)$, $O_{(T_i,k)}$ 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: $\text{ref}_{T_i}^V = (A_{(i,j)}, B_{(i,j)}, C_{(i,j)}) = (t_i, H(t_i)^x, A_v^{\alpha_k})$.

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

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

$$e(H(A_{(i,j)}), g_2^x) = e(B_{(i,j)}, g_2) \tag{1}$$

$$e(C_{(i,j)}, g_2) = e(A_v, g_2^{\alpha_k}) \tag{2}$$

Equation (1) 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 (2) checks whether $C_{(i,j)} = A_v^{\alpha_k}$.

Finally, $\mathcal{V}$ verifies whether $c_{(i,j)}$ is the encryption of $B_{(i,j)}$ with the public key $g_1^{\alpha_k^2}$ by checking if the following equation holds:

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

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,

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

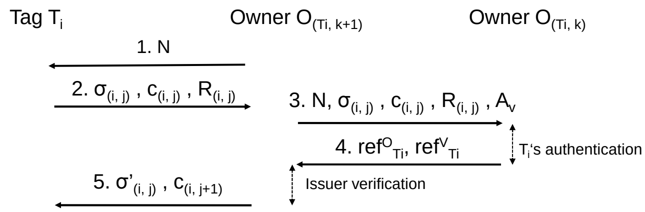

Fig. 4. Ownership transfer in ROTIV

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.

Ownership transfer protocol The setup of the ownership transfer in ROTIV consists of a current owner $O_{(T_i,k)}$, a prospective owner $O_{(T_i,k+1)}$ and a tag T_i as shown in Figure 4. The ownership transfer consists of: **a)** a mutual authentication between T_i and $O_{(T_i,k+1)}$, **b)** an exchange of verification references $\text{ref}_{T_i}^V$ between $O_{(T_i,k)}$ and $O_{(T_i,k+1)}$ to perform issuer verification, and **c)** an exchange of ownership references $\text{ref}_{T_i}^O$ between $O_{(T_i,k)}$ and $O_{(T_i,k+1)}$ to allow $O_{(T_i,k+1)}$ authentication.

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

1. The 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 random number $R_{(i,j)}$ and HMAC $\sigma_{(i,j)}$.

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, $R_{(i,j)}$, $c_{(i,j)}$, $\sigma_{(i,j)}$ and A_v to T_i's owner $O_{(T_i,k)}$.
Given N, $R_{(i,j)}$, $c_{(i,j)}$ and $\sigma_{(i,j)}$, $O_{(T_i,k)}$ 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, $O_{(T_i,k)}$ supplies $O_{(T_i,k+1)}$ with: $\text{ref}_{T_i}^O = (k_i^{\text{old}}, k_i^{\text{new}}, x_i, y_i) = (k_{(i,j)}, k_{(i,j)}, t_i, h_i^x = H(t_i)^x)$ and $\text{ref}_{T_i}^V = (A_{(i,j)}, B_{(i,j)}, C_{(i,j)}) = (t_i, h_i^x, A_v^{\alpha_k})$.

4. Provided with $\text{ref}_{T_i}^O$, $O_{(T_i,k+1)}$ checks if the equation $\sigma_{(i,j)} = \text{HMAC}_{k_{(i,j)}}(N, R_{(i,j)}, c_{(i,j)})$ holds. If it does, this implies that the key $k_{(i,j)}$ provided by $O_{(T_i,k)}$ corresponds to tag T_i.
Given $\text{ref}_{T_i}^V$, $O_{(T_i,k+1)}$ 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, $O_{(T_i,k+1)}$ adds the entry $(y_i, \text{ref}_{T_i}^O)$ into his database $\mathcal{D}_{k+1}$, and finishes 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:

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

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

5. $O_{(T_i,k+1)}$ sends $c_{(i,j+1)}$ and $\sigma'_{(i,j)}$ 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 $\sigma'_{(i,j)}$, T_i authenticates $O_{(T_i,k+1)}$. If the authentication succeeds T_i updates its state accordingly.

5 Privacy and security models

We assume that the communication channel between owners during an ownership transfer and an owner and a verifier during an issuer verification protocol are secure. That is, an adversary $\mathcal{A}$ has only access to the interactions between tags and owners and the wireless interactions between tags and verifiers.

5.1 Privacy

Inspired by previous work on ownership transfer[12, 5], we formally define using experiments the two major privacy requirements for ownership transfer which are *tag forward unlinkability* and *tag backward unlinkability.* In the setting of *tag ownership transfer*, forward unlinkability ensures that when a *new* owner $O_{(T,k+1)}$ acquires T's secrets after a successful ownership transfer at time $k+1$, he still cannot tell whether T has participated in protocol runs at time $t < k+1$. On the other hand, backward unlinkability, ensures that when a *previous* owner $O_{(T,k)}$ releases tag's ownership at time $k+1$, he still cannot tell whether T is involved in interactions that occured at time $t > k+1$.

In the remainder of this section, we assume that the adversary $\mathcal{A}$ has access to 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 queried with two tags T_0 and T_1, randomly chooses $b \in \{0,1\}$ and returns T_b.

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

Forward unlinkability The forward unlinkability experiment captures the capabilities of adversary $\mathcal{A}$ who is allowed to own a tag T at the *end* of his attack, and who has to decide if T was already involved in *previous* interactions.

As discussed in Section 3, in order to achieve constant time authentication and denial of service resistance, we assume that there is at least one communication between T and its owner that is un-observed by $\mathcal{A}$.

Our forward unlinkability experiment is indistinguishability based as proposed by Juels and Weis [8]. Adversary $\mathcal{A}(r,s,t,\epsilon)$ has access to tags in two phases. In the learning phase, as depicted in Algorithm 1, oracle $\mathcal{O}_{\mathcal{T}}$ gives $\mathcal{A}$ two tags T_0 and T_1 that he can eavesdrop on by calling $\textsc{ObserveInteraction}(T_i)$ for a maximum of t times. Note that $\textsc{ObserveInteraction}(T_i\ )$ eavesdrops 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. The ownership of T'_i is then transferred to $\mathcal{A}$ through $\textsc{TransferOwnership}(T'_i, O_{(T'_i,k)}, \mathcal{A})$. $\mathcal{A}$ is now allowed to run up to s mutual authentication with T'_i.

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 $\mathcal{A}$. Then, the oracle $\mathcal{O}_{\text{flip}}$ queried with the tags T_0 and T_1, selects randomly $b \in \{0, 1\}$ and returns the tag T_b to $\mathcal{A}$. Then, the ownership of tag T_b will be transferred to $\mathcal{A}$. Then, $\mathcal{A}$ can run up to t mutual authentication with tag T_b.

$\mathcal{A}$ calls as well oracle $\mathcal{O}_\mathcal{T}$ that supplies him with r tags T''_i. Then, the ownership of T''_i is transferred to $\mathcal{A}$, who now can run up to s mutual authentication with T''_i. Finally, $\mathcal{A}$ outputs his guess of the value of b.

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

```
T_0 ← O_T;
T_1 ← O_T;
for j := 1 to t do
  | ObserveInteraction(T_0);
  | ObserveInteraction(T_1);
end
for i := 1 to r do
  | T'_i ← O_T;
  | TransferOwnership(T'_i, O_(T'_i,k), A);
  |
  | for j := 1 to s do
  |   | RunAuth(T'_i, A)
  | end
end
```

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

```
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};
TransferOwnership(T_b, O_(T_b,k), A);

for j := 1 to t do
  | RunAuth(T_b, A);
end
for i := 1 to r do
  | T''_i ← O_T;
  | TransferOwnership(T''_i, O_(T''_i,k), A);
  |
  | for j := 1 to s do
  |   | RunAuth(T''_i, A)
  | end
end
Output b;
```

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

Definition 3 (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.*

Backward unlinkability Note that 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, see Paise and Vaudenay [15]. In order to achieve at least a slightly weaker notion of backward unlinkability, we add the assumption that a previous owner $O_{(T,k)}$ of tag T *cannot continuously* monitor T after releasing T's ownership. This has been previously suggested by, e.g., Lim and Kwon [12], Dimitrou [5].

The backward unlinkability experiment captures the capabilities of an adversary $\mathcal{A}$ who *releases* the ownership of tag T *during* his attack and has to tell whether T is involved in future protocol transactions.

In the learning phase, cf., Algorithm 3, oracle $\mathcal{O}_{\mathcal{T}}$ selects randomly two tags T_0 and T_1. Then, the ownership of these two tags is transferred to $\mathcal{A}$. $\mathcal{A}$ is allowed to run up to t mutual authentications with tags T_0 and T_1.

$\mathcal{O}_{\mathcal{T}}$ gives $\mathcal{A}$ also a set of r tags T_i'. Then, the ownership of tags T_i' is transferred to $\mathcal{A}$, who can then perform up to s mutual authentications with tags T_i'.

At the end of the learning phase, the oracle $\mathcal{O}_{\mathbb{O}}$ supplies $\mathcal{A}$ with two randomly selected owners. $\mathcal{A}$ then, releases the ownership of tags T_0 and T_1.

```
T_0 ← O_T;
T_1 ← O_T;
TransferOwnership(T_0, O_(T_0,k), A);

TransferOwnership(T_1, O_(T_1,k), A);

for j := 1 to t do
    RunAuth(T_0, A);
    RunAuth(T_1, A);
end
for i := 1 to r do
    T'_i ← O_T;
    TransferOwnership(T'_i, O_(T'_i,k), A);

    for j := 1 to s do
        RunAuth(T'_i, A)
    end
end
O_(T_0,k+1) ← O_O;
TransferOwnership(T_0, A, O_(T_0,k+1));

O_(T_1,k+1) ← O_O;
TransferOwnership(T_1, A, O_(T_1,k+1));
```

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

```
RunAuth(T_0, O_(T_0,k+1));
// Unobserved by A.
RunAuth(T_1, O_(T_1,k+1));
// Unobserved by A.
T_b ← O_flip{T_0, T_1};
for j := 1 to t do
    ObserveInteraction(T_b);
end
for i := 1 to r do
    T''_i ← O_T;
    TransferOwnership(T''_i, O_(T''_i,k), A);

    for j := 1 to s do
        RunAuth(T''_i, A)
    end
end
Output b;
```

Algorithm 4: $\mathcal{A}$'s backward unlinkability challenge 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 $\mathcal{A}$. The oracle $\mathcal{O}_{\text{flip}}$ queried with tags T_0 and T_1, chooses randomly $b \in \{0, 1\}$ and returns the tag T_b to $\mathcal{A}$. $\mathcal{A}$ is allowed to eavesdrop on T_b for a maximum of t times.

$\mathcal{A}$ queries also the oracle $\mathcal{O}_{\mathcal{T}}$ that supplies $\mathcal{A}$ with r tags T_i''. The ownership of T_i'' is transferred to $\mathcal{A}$, who is allowed to run up to s mutual authentication with T_i''. Finally, $\mathcal{A}$ outputs his guess of the value of b. $\mathcal{A}$ is *successful*, if his guess of b is correct.

Definition 4 (Backward Unlinkability). *ROTIV provides backward unlinkability* $\Leftrightarrow$ *For any adversary* $\mathcal{A}$*, inequality* $Pr(\mathcal{A} \text{ is successful}) \leq \frac{1}{2} + \epsilon$ *holds, where* ϵ *is negligible.*

5.2 Security

As ROTIV consists of two main protocols, an ownership transfer protocol and an issuer verification protocol, we introduce the security requirements for each protocol separately. The adversary $\mathcal{A}$ in this section is a direct adaptation of the non-narrow destructive adversary by Vaudenay [18] and Paise and Vaudenay [15] to tag ownership transfer in supply chains.

Ownership transfer A secure ownership transfer must assure the following properties:

a) Mutual authentication A secure ownership transfer protocol must ensure that, when a tag T runs a successful mutual authentication with owner O, this implies that O is T's current owner with high probability. Also, when an owner O runs a successful mutual authentication with a tag T, it yields that T is a legitimate tag with high probability.

We define an authentication game in accordance with Lim and Kwon [12], Vaudenay [18] and Paise and Vaudenay [15]. This game proceeds in two phases. During the learning phase as depicted in Algorithm 5, an adversary $\mathcal{A}(r, s, t, \epsilon)$ is supplied 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 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 s authentications with T_c while impersonating $O_{(T_c,k)}$, (cf., RunAuth(T_c, $\mathcal{A}$)). Also he can start t authentications with $O_{(T_c,k)}$ while impersonating T_c, cf., RunAuth($\mathcal{A}$, $O_{(T_c,k)}$).

```
Tc ← O_T;
for i = 1 to r do
    RunAuth(Tc, O_(Tc,k));
    AlterAuth(Tc, O_(Tc,k));
end
for i = 1 to s do
    RunAuth(Tc, A);
end
for i = 1 to t do
    RunAuth(A, O_(Tc,k));
end
```

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

```
RunAuth(Tc, A);
Tc outputs b_Tc;
RunAuth(A, O_(Tc,k));
O_(Tc,k) outputs b_O_(Tc,k);
```

Algorithm 6: $\mathcal{A}$'s authentication challenge 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, $\mathcal{A}(r, s, t, \epsilon)$ interacts with T_c and initiates an authentication protocol run to impersonate $O_{(T_c,k)}$, cf., $\textsc{RunAuth}(T_c, \mathcal{A})$. 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., $\textsc{RunAuth}(\mathcal{A}, O_{(T_c,k)})$. 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.

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

Definition 5 (Authentication). *ROTIV is secure with regard to authentication ⇔ For any adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ is successful$) \leq \epsilon$ holds, where ϵ is negligible.*

b) Exclusive ownership It ensures that an adversary $\mathcal{A}$ who does not have T's ownership references noted ref_T^O, cannot transfer the ownership of T, unless he rewrites the content of T.

In the learning phase as shown in Algorithm 7, the oracle $\mathcal{O}_{\mathcal{T}}$ supplies $\mathcal{A}(r, s, t, \epsilon)$ with r tags T_i, then, the ownership of tag T_i is transferred to $\mathcal{A}$. $\mathcal{A}$ can run up to s successful mutual authentications with T_i, cf., $\textsc{RunAuth}(T_i, \mathcal{A})$. He can as well at the end of the learning phase, transfer the ownership of tag T_i to an owner O_i selected randomly from the set of owners $\mathbb{O}$.

```
for i := 1 to r do
    T_i ← O_T;
    TransferOwnership(T_i, O_(T_i,k), A);

    for j := 1 to s do
        RunAuth(T_i, A);
    end
    O_i ← O_O;
    TransferOwnership(T_i, A, O_i);
end
```

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

```
T_c ← O_T;
for j := 1 to t do
    s_(T_c,j) := ReadState(T_c);
    ObserveInteraction(T_c);
end
O_c ← O_O;
TransferOwnership(T_c, A, O_c);
O_c outputs b;
```

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

In the challenge phase, cf., Algorithm 8, the oracle $\mathcal{O}_{\mathcal{T}}$ gives $\mathcal{A}(r, s, t, \epsilon)$ a challenge tag T_c.

$\mathcal{A}$ can read T_c's internal state, cf., $\textsc{ReadState}(T_c)$, and eavesdrop on T_c's up to t times. However, $\mathcal{A}$ is not allowed to alter T_c's internal state. 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. $\mathcal{A}$ runs an ownership transfer protocol for T_c with O_c. O_c outputs a bit $b = 1$, if the ownership transfer was successful, and $b = 0$ otherwise. $\mathcal{A}$ is *successful*, if $b = 1$.

Definition 6 (Exclusive ownership). *ROTIV provides exclusive ownership ⇔ For any adversary $\mathcal{A}$, inequality $Pr(\mathcal{A}$ is successful$) \leq \epsilon$ holds, where ϵ is negligible.*

Issuer verification The security of issuer verification ensures that when a verifier $\mathcal{V}$ outputs that the issuer of tag T is $\mathcal{I}$, it implies that $\mathcal{I}$ is the issuer of T 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 gives $\mathcal{A}$ a total of r random tags T_i. The ownership of T_i is then transferred to $\mathcal{A}$, cf. TransferOwnership($O_{(T_i,k)}, \mathcal{A}, T_i$). $\mathcal{A}$ can run up to s mutual authentications with 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 randomly selected from the set of owners $\mathbb{O}$.

```
for i := 1 to r do
    T_i ← O_T;
    TransferOwnership(O_(T_i,k), A, T_i);

    for j := 1 to s do
        RunAuth(T_i, A);
        Verify(T_i,A, V);
    end
    O_i ← O_𝕆;
    TransferOwnership(T_i, A, O_i);
end
```

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

```
CreateTag T_c;
ModifyState(T_c, s'_{T_c});
Verify (T_c, A, V);
V outputs b;
```

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

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 given to $\mathcal{A}$ in the learning phase.

Definition 7 (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.*

6 Privacy analysis

6.1 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 $\mathcal{A}(r, s, t, \epsilon)$ who succeeds in the forward unlinkability experiment with a non negligible advantage ϵ. We will now construct an adversary $\mathcal{A}'(\frac{\epsilon}{2})$, who uses $\mathcal{A}$ as a subroutine, and breaks the DDH assumption in $\mathbb{G}_1$, therewith contradicting the SXDH assumption.

Let $\mathcal{O}_{\text{DDH}}$ be an oracle that selects elements $\alpha, \beta \in \mathbb{F}_q$. Furthermore, $\mathcal{O}_{\text{DDH}}$ 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. $\mathcal{O}_{\text{DDH}}$ 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 public key g_2^x, and an owner O whose public key is g_1^α. the challenge tags T_0 and T_1 stores a ciphertext $c_{(i,j)} = (g_1^{\beta r_{(i,j)}}, h_i^x g_1^{\gamma r_{(i,j)}}), i \in \{0,1\}$ in the learning phase. To break DDH, $\mathcal{A}'$ stores in T_b in the challenge phase, a ciphertext $c_{(i,j+1)} = (g_1^{r_{(i,j+1)}}, h_i^x g_1^{\alpha^{r_{(i,j+1)}}})$.

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 $\mathcal{O}_{\text{DDH}}$ 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.
  1) To issue a tag $T_i, 2 \leq i \leq n-1$ 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 $pk = (g_1^\alpha, g_2^r)$, where r is selected randomly in $\mathbb{F}_q$.
     Note that given the DDH assumption in $\mathbb{G}_2$, $\mathcal{A}$ cannot distinguish g_2^r from $g_2^{\sqrt{\alpha}}$. Therefore, from the point of view of $\mathcal{A}'$ the public key $pk = (g_1^\alpha, g_2^r)$ is valid.
     Also, $\mathcal{A}'$ cannot compute the secret key $sk = \sqrt{\alpha}$ of O. Still, $\mathcal{A}'$ can successfully simulate O: as $\mathcal{A}'$ knows the symmetric keys shared with tags, $\mathcal{A}'$ can compute the HMAC and authenticate tags. $\mathcal{A}'$ can successfully transfer tags' ownership. Note that for each tag T_i $\mathcal{A}'$ can provide **1)** valid verification references: $\text{ref}_{T_i}^V = (t_i, H(t_i)^x, A_v^r)$ which verifies equations (1) and (2). **2)** Valid ownership references $\text{ref}_{T_i}^V = (t_i, H(t_i)^x, k_i^{\text{old}}, k_i^{\text{new}})$.
  2) To issue tags $T_i, i \in \{0,1\}$, $\mathcal{A}'$ randomly selects $r_{(i,0)}$ and $k_{(i,0)} \in \mathbb{F}_q$, computes $h_i = H(t_i)$, and $c_{(i,0)} = (u_{(i,0)}, v_{(i,0)}) = (g_1^{\beta r_{(i,0)}}, h_i^x g_1^{\gamma r_{(i,0)}})$. Finally, $\mathcal{A}'$ stores $s_{(i,0)} = (k_{(i,0)}, c_{(i,0)})$ in tag T_i.
- In the learning phase of the forward unlinkability experiment, $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and gives $\mathcal{A}$ two tags T_0 and T_1.
- $\mathcal{A}$ can eavesdrop on T_0 and T_1 a total of t times. $\mathcal{A}'$ provides $\mathcal{A}$ with r tags T_i'. The ownership of tags T_i' is transferred to $\mathcal{A}$ who can run up to s mutual authentications with T_i'.

- 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 $c_{(i,j+1)} = (g_1^{r_{(i,j+1)}}, h_i^x g_1^{\alpha r_{(i,j+1)}})$.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\text{flip}}$ and transfers the ownership of T_b to $\mathcal{A}$.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and supplies $\mathcal{A}$ with r tags T_i''. Again, the ownership of tags T_i'' is transferred to $\mathcal{A}$, who is allowed to run up to s mutual authentications with T_i''.
- Given that $\mathcal{A}$ does not have access to $N_i, i \in \{0, 1\}$, $k_{(i,j+1)} = G(k_{(i,j)}, N_i)$ 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 ciphertext $c_{(b,j+1)} = (g_1^{r_{(b,j+1)}}, h_b^x g_1^{\alpha r_{(b,j+1)}})$ corresponds to a re-encryption of $c_{(b,j)}$ and therefore to a valid state of tag T_b.

Therefore, $\mathcal{A}$ can output a correct guess for tag 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.

6.2 Backward unlinkability

Theorem 2 (Backward unlinkability). *ROTIV provides backward unlinkability under the SXDH assumption.*

Proof. 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.

Rationale The idea of the proof is to build a ROTIV system with an issuer $\mathcal{I}$ of public key g_2^x, and 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 $\mathcal{O}_{\text{DDH}}$ 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. For simplicity we assume that $ROTIV$ consists of one single owner O whose public key is $pk = (g_1^\alpha, g_2^r)$, and r is selected randomly in $\mathbb{F}_q$.
  Note that given the DDH assumption in $\mathbb{G}_2$, $\mathcal{A}$ cannot distinguish g_2^r from $g_2^{\sqrt{\alpha}}$. Therefore, from the point of view of $\mathcal{A}'$ the public key $pk = (g_1^\alpha, g_2^r)$ is valid.
- To issue a tag $T_i, 0 \leq i \leq n-1$ 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.
- $\mathcal{A}'$ cannot compute the secret key $sk = \sqrt{\alpha}$ of O. Still, $\mathcal{A}'$ can successfully simulate O: as $\mathcal{A}'$ knows the symmetric keys shared with tags, $\mathcal{A}'$ can compute the HMAC and authenticate tags. $\mathcal{A}'$ can successfully transfer tags' ownership. Note that for each tag T_i $\mathcal{A}'$ can provide **1)** valid verification references: $\text{ref}_{T_i}^V = (t_i, H(t_i)^x, A_v^r)$ which verifies equations (1) and (2). **2)** Valid ownership references $\text{ref}_{T_i}^V = (t_i, H(t_i)^x, k_i^{\text{old}}, k_i^{\text{new}})$.
- In the learning phase of the backward unlinkability experiment, $\mathcal{A}'$ simulates $\mathcal{O}_\mathcal{T}$ and gives $\mathcal{A}$ two tags T_0 and T_1.
- The ownership of tags T_0 and T_1 is transferred to $\mathcal{A}$. Adversary $\mathcal{A}$ now can run up to t mutual authentications with T_0 and T_1.
- $\mathcal{A}'$ simulates $\mathcal{O}_\mathcal{T}$ and provides $\mathcal{A}$ with r tags T_i'. The ownership of tags T_i' is transferred to $\mathcal{A}$ who can run up to s mutual authentications with T_i'.
- At the end of the challenge phase, $\mathcal{A}$ transfers the ownership of tag T_0 and T_1 to owner O.
- In the challenge phase, $\mathcal{A}'$ simulating O 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 $\mathcal{O}_{\text{flip}}$ and provides $\mathcal{A}$ with tag T_b.
- $\mathcal{A}'$ simulates $\mathcal{O}_\mathcal{T}$ and supplies $\mathcal{A}$ with r tags T_i''. Again, the ownership of tags T_i'' is transferred to $\mathcal{A}$, who is allowed to run up to s mutual authentications with T_i''.
- Given that $\mathcal{A}$ does not have access to $N_i, i \in \{0,1\}$, $k_{(i,j+1)} = G(k_{(i,j)}, N_i)$ 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 ciphertext $c_{(b,j+1)} = (g_1^\beta, h_b^x g_1^\gamma)$ corresponds to a valid state of tag T_b. Therefore, $\mathcal{A}$ can output a correct guess for tag 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., $\frac{1}{2}$.

Thus, as in the proof above, if $\mathcal{A}$ has a non-negligible advantage ϵ in breaking ROTIV, $\mathcal{A}$ will have a non negligible advantage $\epsilon' = \frac{\epsilon}{2}$ in breaking DDH. This leads to a contradiction.

7 Security Analysis

7.1 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 satisfies 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 $\mathcal{A}'(p, \epsilon)$ can choose p messages $m_1, \ldots, m_p$, 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 $m \neq m_i, 1 \leq i \leq p$, is negligible.

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

Proof. To simplify the proof, we assume that the key k 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 $\mathcal{A}(r, s, t, \epsilon)$ is able to break the security of the authentication scheme with non-negligible advantage, then we can construct adversary $\mathcal{A}'(p, \epsilon')$ that breaks the resistance to existential forgery of HMAC with non-negligible advantage $\epsilon' = \epsilon$.

Let $\epsilon = \epsilon_{T_c} + \epsilon_{O_{(T_c,k)}}$ such that: ϵ_{T_c} is $\mathcal{A}$'s advantage in impersonating T_c, and $\epsilon_{O_{(T_c,k)}}$ is $\mathcal{A}$'s advantage in impersonating T_c's owner $O_{(T_c,k)}$.

Rationale To break the existential forgery of an HMAC of secret key k, $\mathcal{A}'$ simulates both the challenge tag T_c and the owner of T_c called $O_{(T_c,k)}$, where T_c and $O_{(T_c,k)}$ share the secret key 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 σ for a message m which he has not seen before. Thus, to break the security of HMAC, $\mathcal{A}'$ answers with the pair (m, σ).

Construction

- $\mathcal{A}'$ simulates issuer $\mathcal{I}$ and creates n 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 $t_i \in \mathbb{F}_q, 1 \leq i \leq n-1$ and computes $h_i = H(t_i)$. Also, $\mathcal{A}'$ 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 $k_i \in \mathbb{F}_q, 1 \leq i \leq n-1$ and stores $s_{(i,0)} = (k_i, c_{(i,0)})$ into $T_i, 1 \leq i \leq n-1$.
  3) To create T_c whose secret key is k, $\mathcal{A}'$ stores $s_{(T_c,0)} = c_{(T_c,0)}$. To compute the HMAC during the authentication, T_c does not use directly the secret key k but instead queries the oracle $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and returns T_c to $\mathcal{A}$.
- In the learning phase, $\mathcal{A}'$ starts r mutual authentications with T_c that $\mathcal{A}$ can eavesdrop on. $\mathcal{A}'$ as well starts another r mutual authentications that $\mathcal{A}$ can alter by injecting fake messages up to r times. $\mathcal{A}$ can start s authentications with T_c while impersonating T_c's owner $O_{(T_c,k)}$. He can also start t authentications with $O_{(T_c,k)}$ while impersonating T_c.
- $\mathcal{A}'$ simulates both T_c and $O_{(T_c,k)}$.
  - $\mathcal{A}'$ simulates T_c. When T_c receives the first message of the authentication protocol which is a random nonce N_j:
    1) $\mathcal{A}'$ generates a random number R_j.
    2) $\mathcal{A}'$ queries the oracle, $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with $m_j = (N_j, R_j, c_{(T_c,j)})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\sigma_j = \text{HMAC}_k(m_j)$.
    3) $\mathcal{A}'$ sends R_j, $c_{(T_c,j)}$ and σ_j to $O_{(T_c,k)}$.
  - $\mathcal{A}'$ simulates $O_{(T_c,k)}$. When $O_{(T_c,k)}$ receives the second message of the authentication protocol, that is $(R_j, c_{(T_c,j)}, \sigma_j)$: **1)** $\mathcal{A}'$ identifies T_c by decrypting $c_{(T_c,j)}$, if the identification fails, then $\mathcal{A}'$ aborts the authentication.
    Otherwise, **2)** $\mathcal{A}'$ queries the oracle with message $m_j = (N_j, R_j, c_{(T_c,j)})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\text{HMAC}_k(m_j)$.
    3) $\mathcal{A}'$ checks whether $\sigma_j = \text{HMAC}_k(m_j)$, if not, then $\mathcal{A}'$ aborts authentication.
    Otherwise, **4)** $\mathcal{A}'$ computes $c_{(T_c,j+1)}$ and queries $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with message $m'_j = (R, c_{(T_c,j+1)})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\sigma'_j = \text{HMAC}_k(m'_j)$.
    5) $\mathcal{A}'$ sends the last message of authentication $(c_{(T_c,j+1)}, \sigma'_j)$ to T_c.
  - $\mathcal{A}'$ simulates T_c. When T_c receives the last message of authentication $(c_{(T_c,j+1)}, \sigma'_j)$: **1)** $\mathcal{A}'$ queries $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ with $m'_j = (R, c_{(T_c,j+1)})$ and $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\text{HMAC}_k(m'_j)$.
    2)$\mathcal{A}'$ checks whether $\sigma'_j = \text{HMAC}_k(m'_j)$. If not, $\mathcal{A}'$ aborts the authentication. Otherwise, T_c updates its stored ciphertext to $c_{(T_c,j+1)}$.
- 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 $m = (N, R, c_{(T_c,j')})$. $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$ returns $\sigma = \text{HMAC}_k(m)$.

Finally, $\mathcal{A}'$ sends R, $c_{(T_c,j')}$ and σ to $\mathcal{A}$.
$\mathcal{A}$ replies with $(c_{(T_c,j'+1)}, \sigma')$, such that $\sigma' = \text{HMAC}_k(m')$ and $m' = (R, c_{(T_c,j'+1)})$ with advantage $\epsilon_{O_{(T_c,k)}}$.
To break the existential forgery of HMAC, $\mathcal{A}'$ simply outputs (m', σ').

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 and sends R, a ciphertext $c_{(T_c,j')}$ and σ to $\mathcal{A}'$. Note that $\sigma = \text{HMAC}_k(m)$ where $m = (N, R, c_{(T_c,j')})$, with advantage ϵ_{T_c}.
To break the existential forgery of HMAC_k, $\mathcal{A}'$ outputs (m, σ).

Now, we quantify $\mathcal{A}$'s advantage. $\mathcal{A}'$ succeeds in breaking the existential forgery of HMAC

1) If $\mathcal{A}'$ makes at most $l = 4r + s + t + 1$ calls to $\mathcal{O}^{\text{forge}}_{\text{HMAC}_k}$.
2) With advantage $\epsilon_{(O_{T_c},k)}$, if $\mathcal{A}$ impersonates $O_{(T_c,k)}$ in the challenge phase.
3) With advantage ϵ_{T_c}, if $\mathcal{A}$ impersonates T_c in the challenge phase.

Let p denotes the probability that $\mathcal{A}$ impersonates T_c. Hence, $\mathcal{A}$'s advantage is: $\epsilon' = (1-p)\ \epsilon_{O_{(T_c,k)}} + p\ \epsilon_{T_c} \leq \epsilon$. Therefore, if $\mathcal{A}$'s advantage ϵ in breaking ROTIV's security is non-negligible, $\mathcal{A}'$ will be able to break the existential forgery of HMAC with a non-negligible advantage ϵ. This leads to a contradiction under the security of HMAC.

7.2 Exclusive ownership

Theorem 4 (Exclusive Ownership). *The ownership transfer protocol in ROTIV provides exclusive ownership under the security of hash function H.*

Proof. Assume there is an adversary $\mathcal{A}(r, s, t, \epsilon)$ who succeeds in the exclusive ownership experiment 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 $H(t)$. $\mathcal{A}'$ breaks the one wayness of H, if given $H(t)$, he outputs t 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_c. $\mathcal{A}'$ creates a tag T_c such that $s_{(0,c)} = (k_{(0,c)}, c_{(0,c)})$, where $c_{(0,c)} = (g_1^{r_{(0,c)}}, h_c^x g_1^{\alpha_k^2 r_{(0,c)}})$. If $\mathcal{A}$ has a non negligible advantage 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, $\text{ref}^O_{T_c} = (t_c, h_c^x, k_{\text{old}}, k_{\text{new}})$, where $h_c = H(t_c)$.
To break H's one wayness, $\mathcal{A}'$ outputs t_n.

Construction

– $\mathcal{A}'$ simulates the issuer $\mathcal{I}$ and creates n tags T_i, $1 \leq i \leq n$.
  1) $\mathcal{A}'$ selects randomly $x \in \mathbb{F}_q$. Here x will be the secret key of the issuer.
  2) For each tag $T_i, 1 \leq i \leq n-1$, $\mathcal{A}'$ 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. $\mathcal{A}'$ outputs $\mathrm{ref}_{T_i}^{O} = (t_i, h_i^x, k_{(i,0)}, k_{(i,0)})$.
  3) Finally, he creates tag T_c. $\mathcal{A}'$ queries $\mathcal{O}_H$ that returns hash h_c. $\mathcal{A}'$ selects a random number $r_{(c,0)}$ and computes $c_{(c,0)} = (u_{(c,0)}, v_{(c,0)}) = (g_1^{r_{(c,0)}}, h_c^x g_1^{\alpha_k^2 r_{(c,0)}})$. Therewith, $\mathcal{A}'$ selects randomly $k_{(c,0)} \in \mathbb{F}_q$ and stores $s_{(c,0)} = (k_{(c,0)}, c_{(c,0)})$ into tag T_c.
– $\mathcal{A}$ enters the learning phase. $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$, i.e., $\mathcal{A}'$ supplies $\mathcal{A}$ with r tags. $\mathcal{A}'$ selects randomly a tag T_i from the n tags he created and checks whether $T_i = T_c$. If so, $\mathcal{A}'$ stops the experiment, otherwise, $\mathcal{A}'$ supplies $\mathcal{A}$ with tag T_i, and transfers T_i's ownership to $\mathcal{A}$ using the ownership references $\mathrm{ref}_{T_i}^{O} = (t_i, h_i^x, k_{(i,0)}, k_{(i,0)})$.
– $\mathcal{A}$ can run up to s mutual authentications with T_i.
– At the end of the learning phase, $\mathcal{A}$ transfers the ownership of tag T_i to an owner from the set of legitimate owners.
– In the challenge phase, $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and selects randomly a tag T. If $T \neq T_c$, $\mathcal{A}'$ stops the experiment. Otherwise, $\mathcal{A}'$ provides $\mathcal{A}$ with T_c.
– $\mathcal{A}$ now can read T_c's internal state and he eavesdrops on T_c for a maximum of t times.
– $\mathcal{A}'$ 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 supply O_c during the ownership transfer protocol with $\mathrm{ref}_{T_c}^{O} = (t_c, h_c^x, k_{\mathrm{old}}, k_{\mathrm{new}})$, where $h_c = H(t_c)$.
  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_n in the learning phase, and choosing T_n in the challenge phase. The probability that $\mathcal{A}'$ does not choose T_n in the learning phase is $(1 - \frac{1}{n})^r$. The probability that $\mathcal{A}'$ chooses T_n in the challenge phase is $\frac{1}{n}$.

Hence, $\mathcal{A}'$'s advantage is: $\epsilon' = \Pr(\mathcal{A}'$ does not abort the experiment$) \cdot \epsilon = \frac{1}{n}(1 - \frac{1}{n})^r \cdot \epsilon$.

This leads to a contradiction under the security of H.

7.3 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 $\mathcal{A}(r, s, \epsilon)$ 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'.

BCDH assumption: Given $g_1, g_1^x, g_1^y, g_1^z \in \mathbb{G}_1$ and $g_2, g_2^x, g_2^y \in \mathbb{G}_2$, the probability to compute $e(g_1, g_2)^{xyz}$ is negligible.

Let $\mathcal{O}_{\text{BCDH}}$ be an oracle, when queried selects randomly $x, y, z \in \mathbb{F}_q$ and returns $g_1, g_1^x, g_1^y, g_1^z, g_2,$
g_2^x, g_2^y.

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, $\text{ref}_{T_c}^V = (A_c, B_c, C_c) = (t_c, h_c^x, C_c)$, where h_c is the hash of t_c. Therefore, to break the BCDH assumption, $\mathcal{A}'$ simulates the outputs of H as a random oracle 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 $h_c = H(t_c) = g_1^{zr_c}$.

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

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 $\text{coin}(t) \in \{0, 1\}$ such that: $\text{coin}(t) = 1$ with probability p, and is equals to 0 with probability $1 - p$. To compute $H(t)$, $\mathcal{A}'$ checks $\text{coin}(t)$: if $\text{coin}(t) = 0$, $\mathcal{A}'$ looks up r_t in T_H, and answers $H(t) = g_1^{r_t}$. Otherwise, if $\text{coin}(t) = 1$, $\mathcal{A}'$ answers with $H(t) = (g_1^z)^{r_t}$.

Construction

- $\mathcal{A}'$ first queries $\mathcal{O}_{\text{BCDH}}$ to receive $g_1, g_1^x, g_1^y, g_1^z \in \mathbb{G}_1$ and $g_2, g_2^x, g_2^y \in \mathbb{G}_2$.
- $\mathcal{A}'$ 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 $\text{coin}(t_i) = 1$ $\mathcal{A}'$ 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, $\text{ref}_{T_i}^O = (k_{(i,0)}, k_{(i,0)}, h_i, h_i^x)$.
- $\mathcal{A}$ enters the learning phase.
- $\mathcal{A}'$ simulates $\mathcal{O}_{\mathcal{T}}$ and supplies $\mathcal{A}$ with r tags T_i. $\mathcal{A}'$ using the ownership references of tag T_i $\text{ref}_{T_i}^O$, transfers the ownership of T_i to $\mathcal{A}$.
- Provided with the ownership references $\mathcal{A}$ has full control of T_i, and he can now run s authentications with T_i and s issuer verification for tag T_i. At the end of the learning phase, $\mathcal{A}$ transfers the ownership of 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 $\text{coin}(t_c)$, if $\text{coin}(t_c) = 0$, $\mathcal{A}'$ stops the experiment. Otherwise, $\mathcal{A}'$ selects randomly $r_c \in \mathbb{F}_q$ and answers with $h_c = (g_1^z)^{r_c} = g_1^{zr_c}$.

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

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

  Finally, to break BCDH, $\mathcal{A}'$ computes

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

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

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

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

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

8 Related work

Molnar et al. [13] address the problem of ownership transfer in RFID systems by using tag pseudonyms and relying on a trusted third party. Here, the TTP is the only entity than can identify tags. To transfer ownership of tag T, the current owner of T, $O_{(T,k)}$, and the prospective owner of T, $O_{(T,k+1)}$, contact the TTP. who then, provides $O_{(T,k+1)}$ with T's identity. Once the ownership transfer of T takes place, the TTP refuses identity requests from T's previous owner $O_{(T,k)}$. However, relying on a TTP is a drawback: in many scenarios, the availability of a trusted third party during tag ownership transfer is probably unrealistic.

Other solutions based on symmetric primitives have been proposed by Lim and Kwon [12], Fouladgar and Afifi [6], Song [17], and Kulseng et al. [10]. These schemes however suffer as discussed in section 2.2 from three major drawbacks: **1.)** tag identification and authentication is linear in the number of tags, **2.)** desynchronization and **3.)** no tag issuer verification.

Kapoor and Piramuthu [9] suggests a two party ownership transfer protocol based on keyed hash functions. In order to provide forward unlinkability, the new owner of tag T, $O_{(T,k+1)}$ does not have access to the key of the previous owner $O_{(T,k+1)}$. Also, to cope with desynchronization, T's owner does not update the shared key unless he receives an acknowledgment from T. However, as the scheme relies on symmetric primitives it still suffers from linear time authentication and lack of issuer verification.

Dimitrou [5] proposes a solution to ownership transfer that relies on symmetric cryptography while relaxing the privacy requirements for both backward and forward unlinkability. Unlike previous schemes on ownership transfer, this solution allows an owner of a tag to revert the tag to its original state. This is useful for after sales services where a retailer can recognize a sold tag T. Note that ROTIV offers the same feature: a tag T's unique identifier will allow any owner to verify whether he owned T before or not.

9 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. 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.

[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] 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.

[9] G. Kapoor and S. Piramuthu. Single RFID Tag Ownership Transfer Protocols. *IEEE Transactions on Systems, Man, and Cybernetics*, Issue 99:1–10, 2011. ISSN 1094-6977.

[10] L. Kulseng, Z. Yu, Y. Wei, and Y. Guan. Lightweight mutual authentication and ownership transfer for rfid systems. In *INFOCOM*, pages 251–255, 2010.

[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 Peng Ning, Sihan Qing, and Ninghui Li, editors, *International Conference on Information and Communications Security – ICICS'06*, volume 4307 of *Lecture Notes in Computer Science*, pages 1–20, Raleigh, North Carolina, USA, December 2006. Springer.

[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] M. Ohkubo, K. Suzuki, and S. Kinoshita. Cryptographic Approach to "Privacy-Friendly" Tags. In *RFID Privacy Workshop*, MIT, Massachusetts, USA, November 2003.

[15] 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.

[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.