4-2018

Every step you take, I'll be watching you: Practical StepAuth-entication of RFID paths

Kai BU
Zhejiang University

Yingjiu LI
Singapore Management University, yjli@smu.edu.sg

Published in IEEE Transactions on Information Forensics and Security, 2017 October, Pages 1-16
https://doi.org/10.1109/TIFS.2017.2768022

Every Step You Take, I'll Be Watching You: Practical STEPAUTH-entication of RFID Paths

Kai Bu*, *Member, IEEE*, Yingjiu Li, *Member, IEEE*

***Abstract*—Path authentication thwarts counterfeits in RFID-based supply chains. Its motivation is that tagged products taking invalid paths are likely faked and injected by adversaries at certain supply chain partners/steps. Existing solutions are path-grained in that they simply regard a product as genuine if it takes any valid path. Furthermore, they enforce distributed authentication by offloading the sets of valid paths to some or all steps from a centralized issuer. This not only imposes network and storage overhead but also leaks transaction privacy.**

We present STEPAUTH, the first step-grained path authentication protocol that is practically efficient for authenticating products with strict path bindings. We encode a path into a secret with minimum path visibility disclosure between adjacent steps. Carrying the secret, a product has to go through steps in the exact order as in the designated path to pass authentication. STEPAUTH enforces no tag computation and enables each step to locally verify path secrets without pre-offloaded valid-path sets. Toward an even higher security guarantee, STEPAUTH can hinder an adversary capable of compromising all steps from forging valid secrets. We make STEPAUTH practically efficient by taking advantage of nested encryption and hybrid encryption. To achieve a 128-bit security for a practically long path of 100 steps, STEPAUTH generates a secret around 10 KB, which can be well supported by high-memory EPC Gen2 tags. Such secrets take STEPAUTH less than 1 s to encode and around 10 ms to verify.

***Index Terms*—RFID, path authentication, supply chain management.**

I. INTRODUCTION

RADIO-Frequency Identification (RFID) paths have been widely used to reveal counterfeit products. In RFID-enabled supply chains, tags affixed to products carry secrets related to the paths taken by products. Enroute readers deployed at each step (i.e., a supply chain partner) of a path verify products via authenticating their path secrets. The motivation for path authentication is that injected counterfeits usually take different paths than their authentic counterparts [1]. According to a recent report by OECD and the EU's Intellectual Property Office in April 2016, global trade in counterfeit products worth almost half a trillion dollars a year, accounting for around 2.5% of global imports [2]. The type of counterfeits ranges from cheap eggs [3] to expensive jewelry [4] and from low-end event tickets [5] to high-tech electronic devices [6]. Such counterfeit products may impose various threats on consumers. For example, counterfeit luxuries cause financial loss to consumers while counterfeit foods and electronic devices threat consumers' health and safety. Counterfeit detection based on RFID paths is therefore practically imperative.

A. Related Work

The state of the art for RFID path authentication is quasi-distributed and path-grained. The centralized issuer—knowing all valid paths—offloads them to all or some steps, which act as checkpoints. Enroute readers encode themselves into the path secrets carried by passing tags. Checkpoints authenticate a tag's secret by verifying whether it can be derived by a valid path pertaining to local valid-path sets. The pioneering protocol, Pathchecker [1], uses the path ends as checkpoints and enforces an $\mathcal{O}(n)$ authentication when deriving secrets by one of n valid paths after another to compare with those carried in tags. Moreover, Pathchecker requires tags to evolve secrets, which involves complex cryptography toward security and privacy. This makes it prohibitively computation-intensive for low-cost tags [7]. TRACKER [7] relieves tags from computation by letting readers take over secret verification and update. Cai *et al.* [8] and Wang *et al.* [9] improve upon TRACKER by stronger privacy requirements and more compact secrets, respectively. Mamun *et al.* [10] and Ray *et al.* [11] secure tags against untrusted readers by reader authentication, which again necessitates cryptographic computation requirements to tags. Instead of waiting until the final stage of a supply chain to reveal counterfeits, a two-level path authentication protocol [12] segments a path and uses segment ends as checkpoints. CHECKER [13] further generalizes this idea and enables each reader to verify the validity of the path taken by a tag so far.

However, all these path-grained authentication protocols are not well suited to enforce delivery regulation that requires each product to follow a designated path. Take the build-to-order supply chain management strategy [14] for example. A manufacturer may need to deliver ordered or even customized products to a specific retailer via a series of specified distributors and wholesalers but not simply following any other valid paths down the supply chain hierarchy. A feasible way to adapt path-grained authentication is to augment each reader with product-path bindings, which enable readers to make delivery decision and verify delivery correctness. This, however, incurs frequent issuer-reader communication upon path update[1] or distributing new products. Considering the large volume of products in today's supply chains (e.g., over

K. Bu* is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: kaibu@zju.edu.cn).

Y. Li is with the School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore (e-mail: yjli@smu.edu.sg).

*Corresponding Author: Kai Bu.

EDICS: CYB-CP Cyber-Physical security.

[1]For highly dynamic supply chains where paths are hard to predict, each reader has to query the centralized issuer for online secret verification [15]. As with most related work, we focus on relatively stable supply chains.

TABLE I
COMPARISON OF STEPAUTH WITH PATH-GRAINED AUTHENTICATION PROTOCOLS.

Legend: NTC: No Tag Computation; CP: Checkpoint; NPS: No Path Set MPV: Minimal Path Visibility; SG: Step-grained					
Protocol	NTC	CP	NPS	MPV	SG
Pathchecker [1] Mamun *et al.* [10] Ray *et al.* [11]	✗	path end	✗	✗	✗
TRACKER [7] Cai *et al.* [8] Wang *et al.* [9]	✓	path end	✗	✗	✗
Two-level [12]	✓	segment end	✗	✗	✗
CHECKER [13]	✓	each reader	✗	✗	✗
Proxy Re-signature [17]	✓	path end	✗	✓	✗
STEPAUTH	✓	each reader	✓	✓	✓

480 million products on Amazon US [16]), exporting product-path bindings to enroute readers would cause huge overhead of network bandwidth and reader storage. What might be of more concern to supply chain partners is that revealing valid paths may leak their business strategies. Burbridge *et al.* [17] thus advocate minimal path visibility that limits linkage disclosure between only adjacent readers. It is, however, challenging to implement the adopted proxy re-signature scheme in [17] without a trusted third party [13].

B. Our Solution and Contributions

In this paper, we present STEPAUTH as the first practically efficient protocol for distributed and step-grained path authentication with minimal path visibility. Being step-grained, STEPAUTH enforces a product to follow a succession of ordered steps/readers, which constitute the designated path. The major idea is to encode the designated path into a secret. We use nested encryption to enforce the step order. Using nested encryption, we recursively incorporate a step's secret into that of its previous step. STEPAUTH encrypts path secrets in such a way that they reveal only two reader identifiers to each reader. One is the identifier of the reader itself for verifying delivery correctness and the other one is its next step for making delivery decision. Each enroute reader thus knows only its adjacent readers and guarantees minimal path visibility. To guarantee both security and efficiency, we leverage hybrid encryption that uses symmetric cryptography to efficiently encrypt and decrypt path secrets and uses public key cryptography to encrypt and decrypt the symmetric key. For the public key cryptography, we generate a pair of encryption key k_i^{e} and decryption key k_i^{d} for each reader R_i. A reader's symmetric key k_i^{s} is randomly generated and encrypted using the encryption key k_i^{e} upon secret construction. Neither the issuer nor readers need to store these symmetric keys. Each reader R_i is granted with its decryption key k_i^{d} when it joins the supply chain and registers to the issuer. Readers can thus use decryption keys to get symmetric keys and then authenticate path secrets, without querying the issuer. This guarantees fully distributed authentication. Toward an even higher security guarantee, we further explore techniques to prevent an adversary from forging a valid secret even if the adversary can compromise all readers.

We highlight STEPAUTH's major contributions as follows while comparing STEPAUTH with prior path-grained authentication protocols in Table I.

- We initiate the step-grained path authentication problem. It helps to enforce a practical supply chain management policy that delivers a product along a designated path. Existing path authentication protocols are not well suited to enforce such a policy as they allow a product to take any valid path.
- We present STEPAUTH, the first protocol for step-grained yet distributed path authentication. It encodes paths into secrets and stores them in corresponding tags. Such secrets enable each reader to locally authenticate tags and make delivery decision without querying the issuer.
- STEPAUTH requires no tag computation and thus favors low-cost tags.
- STEPAUTH guarantees minimal path visibility by not offloading valid-path sets to readers.
- STEPAUTH can generate compact path secrets by leveraging nested encryption and hybrid encryption. We investigate implementation of STEPAUTH based on the Elliptic Curve Integrated Encryption Scheme (ECIES) [18] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [19]. Even for a practically long enough path of 100 steps, the generated secrets that achieve a 128-bit security by STEPAUTH are around 10 KB, which can be well supported by high-memory EPC Gen2 tags. For example, Marubeni Chemix and Xerafy have produced tags with 8 KB memory [20] and Fujitsu further pushes the boundary to 64 KB [21]. Based on the Crypto++ 5.6.0 Benchmarks [22], [23], STEPAUTH takes less than 1 s to generate such secrets and around 10 ms to verify them.
- STEPAUTH is practically secure and privacy-preserving. An attacker can breach tag/step unlinkability only if it can compromise readers' decryption keys while it cannot forge a valid secret even if it compromises all readers.

The rest of the paper is organized as follows. Section II models the step-grained path authentication problem. Section III outlines STEPAUTH framework and design challenges. Section IV and Section V detail the design and prove security and privacy, respectively. Section VI explores implementation choices and evaluates performance. Section VII discusses potential limitations and suggests countermeasures. Finally, Section VIII concludes the paper.

II. PATH AUTHENTICATION MADE STEP-GRAINED

In this section, we define the step-grained path authentication problem. For ease of understanding, we follow conventional terms and notions in literature (e.g., [1], [7], [13]). We adopt a simpler supply chain model yet stricter solution requirements toward striving for previous solutions' joint benefits without forcing their sacrifices on efficiency, security, or privacy. In a nutshell, we envision a solution that offers distributed, step-grained path authentication with minimal path visibility disclosure. Being distributed and step-grained, the solution should enable each step on a path to independently verify the path taken by a tag so far. A reader does not need to query a centralized, trusted party upon verification. Unlike existing path-grained solutions, the verification does not accept a tag simply because its taken path pertains to a

set of valid paths (or their beginning segments) preloaded to final (or intermediate) steps. We observe that a tag following a valid path cannot guarantee that the tag follows the correct one. In practical supply chains, it is normal that a manufacturer distributes multiple types of product across different valid paths, but each of which may allow only certain types of product. From the privacy perspective, the solution should also limit minimal path visibility between adjacent steps to, for example, protect business strategy of supply chain partners.

A. Supply Chain Model

A supply chain consists of a series of supply chain partners (e.g., manufacturers, distributors, wholesalers, and retailers) that manufacture and circulate products [24]. It can be modeled as a digraph $G = (V, E)$ [7]. Each vertex $v \in V$ represents a supply chain partner. Each edge $e = \overrightarrow{v_i v_j} \in E$ represents a business strategy allowing partner v_i to deliver products to partner v_j. A path p of l consecutive edges $p = (e_0, ..., e_i, e_{i+1}, ..., e_{l-1}) \in \mathcal{P}$, where $e_i.\text{head} = e_{i+1}.\text{tail}$ and $\mathcal{P}$ represents the set of valid paths, thus enforces a policy on which partners a product can visit. Alternatively, the above path p can be represented as $l+1$ consecutive partners $p = (v_0, ..., v_j, v_{j+1}, ..., v_l)$, where $\overrightarrow{v_j v_{j+1}} \in E$ and v_j is usually called a *step* on the path [1]. We assume that the designated path of a product is known as *a priori*. This resonates with the build-to-order supply chain management strategy, which favors more of customer needs and improves manufacturer competitiveness [14]. Instead of manufacturers aimlessly producing products to sell, they reactively schedule production according to received product orders. For example, if two retailers—A and B—plan to purchase some products from a wholesaler, they need to respectively submit an order to the wholesaler. The orders will be further submitted to a distributor and a manufacturer. Then the manufacturer is aware of the binding of products and paths.

RFID-enabled supply chains leverage RFID technology to implement product tracking [1]. Each product is affixed with an RFID tag carrying product related data. Tag genuineness serves as an important indicator of product authenticity. Deployed RFID readers interact with tags for genuineness verification. The RFID-enabled supply chain we adopt consists of three types of entities, that is, an issuer $\mathcal{I}$, a set $\mathcal{R}$ of readers, and a set $\mathcal{T}$ of tags.

1) ***Issuer*** *$\mathcal{I}$:* The issuer is a manufacturer with the knowledge of product-path couplings. For path authentication, issuer $\mathcal{I}$ generates a secret out of each tag's designated path and loads the path secret to the tag. Path secrets inside tags should vary stepwise to avoid tracking attacks. The issuer needs to accordingly configure readers in such a way that they can verify and update path secrets. Such reader configurations are deemed secure because readers have sufficient storage space and computation resources to support secure wired or wireless communication [7].

2) ***Reader****:* $R_i \in \mathcal{R}$, where $0 \leq i \leq |\mathcal{R}| - 1$. Each step is equipped with a reader to communicate with tags attached to products[2]. Reader-tag communication is against an insecure wireless channel because resource-constrained tags can hardly afford complex cryptography [25]. For distributed path authentication, we require that each reader locally verify path secrets without querying a centralized party with global knowledge of tag-path bindings. To do so, each reader should be pre-granted by the issuer necessary data (e.g., keys) that helps to verify the validity of path secrets. Furthermore, a reader needs also to update path secrets for its next-hop reader, if any, to verify.

3) ***Tag****:* $T_j \in \mathcal{T}$, where $0 \leq j \leq |\mathcal{T}| - 1$. Each tag is attached to a product, whose authenticity is implied by tag genuineness. It features memory space and supports read and write operations by readers. Normally, a tag stores a unique ID as well as product related data. We in this paper focus on path secrets. Given a set of $\mathcal{S}$ valid secrets, let $s^k_{T_j}$ represent the kth secret of tag T_j, where $k \geq 0$. The secret of tag T_j is initialized as $s^0_{T_j}$ by the issuer. If passing a reader's verification, it will be updated by the reader from $s^k_{T_j}$ to $s^{k+1}_{T_j}$.

Based on the preceding definitions, we formalize the following functions featured by the issuer (INITIALIZE) and readers (READ, VERIFY, UPDATE, WRITE) [1], [13].

- INITIALIZE : $\mathcal{T} \times \mathcal{P} \to \mathcal{S}$ generates the initial path secret $s^0_{T_j}$ for tag T_j using its designated path p_{T_j}.
- READ : $\mathcal{T} \to \mathcal{S}$ reads tag T_j's current secret $s^k_{T_j}$.
- VERIFY : $\mathcal{S} \times \mathcal{R} \to \{0, 1\}$ verifies the validity of secret $s^k_{T_j}$ read from tag T_j.

$$\text{VERIFY}(s^k_{T_j}, d^{\text{aux}}_{R_i}) = \begin{cases} 1, \text{ if } T_j\text{'s current step} \\ \quad \text{in } p_{T_j} \text{ is } R_i; \\ 0, \text{ otherwise,} \end{cases} \tag{1}$$

  where $d^{\text{aux}}_{R_i}$ represents auxiliary data of reader R_i needed for verifying $s^k_{T_j}$. For example, if $s^k_{T_j}$ is encrypted with the issuer's secret key, $d^{\text{aux}}_{R_i}$ might contain the issuer's public key for R_i to decrypt $s^k_{T_j}$.
- UPDATE : $\mathcal{S} \times \mathcal{R} \to \mathcal{S}$ updates the secret from $s^k_{T_j}$ to $s^{k+1}_{T_j}$.

$$\text{UPDATE}(s^k_{T_j}, d^{\text{aux}}_{R_i}) = s^{k+1}_{T_j}.$$

- WRITE : $\mathcal{S} \to \mathcal{T}$ writes the updated secret $s^{k+1}_{T_j}$ to tag T_j. For simplicity, we also use WRITE to denote the function for issuer $\mathcal{I}$ to load initial path secrets.

Among these five functions, INITIALIZE, VERIFY, and UPDATE are of more design challenge [1], [13].

B. Adversary Model

We consider a global active adversary $\mathcal{A}$ against a path authentication protocol's security and privacy. Following the assumption by existing path authentication protocols, the adversary can compromise readers but cannot compromise the issuer.

To breach security, adversary $\mathcal{A}$'s primary goal is to forge path secrets that can pass the authentication protocol[3]. Once

[2]We use "step" and "reader" interchangeably whenever no confusion arises.

[3]Note that an adversary may also sabotage path authentication. For example, it can make authentic products fail authentication through writing invalid secrets to their attached tags. Such an attack can be mitigated by lightweight mutual authentication schemes [26]. We, however, focus on an adversary with more incentives to inject counterfeits into a supply chain than to subvert it.

such forged secrets are found, $\mathcal{A}$ can program them to tags and attach the tags to counterfeits, which are likely circulated as authentic ones and endanger potential customers. Adversary $\mathcal{A}$ may conduct various other active attacks to increase the chance of forging valid secrets. For example, $\mathcal{A}$ with physical tampering capability can crack tag memory or even reader memory, acquiring path secrets and keys used to update them.

Adversary $\mathcal{A}$ may also infer tag privacy using acquired data. A global adversary can eavesdrop on tags' communication or launch its other capable attacks at all steps in the supply chain. It might mount a subordinate adversary at each step and then synthesize their acquired data to a centralized one. Although the assumed global adversary overestimates practical attacking resources, it forces us to design a more robust protocol than when we consider a localized adversary attacking only a limited number of steps. Inferred tag privacy includes, for example, which path a tag took [7]. We will shortly introduce specific privacy concerns when reasoning about design requirements for path authentication.

Throughout this paper, we assume that adversary $\mathcal{A}$ is probabilistic polynomial time bounded in terms of a security parameter such as 128 bits.

C. Step-Grained Path Authentication Problem

We advocate fine-grained RFID path authentication with step granularity. Specifically, each product should follow a designated path. Such a tag-path binding yields a practically stricter requirement—an acceptable tag should visit the exact steps in its designated path following the exact order. Previous solutions are, however, path-grained; they accept a tag simply if it takes any path among all valid ones. In many supply chains, not all products can take all valid paths. Take the pharmaceutical supply chain for example [27]. An American pharmaceutical manufacturer might distribute its products nationwide as well as export them to other countries like China. Then packages without specific Chinese instructions on uses, dosage, precautions, and drug interactions should be prohibited from paths toward China, although which are valid paths. Note that it is not practically efficient to simply issuing the bindings of tagged products and their designated paths to steps; we will shortly present the reasons in Section II-D.

1) ***Step-grained path authentication problem****:* Given tag T_j's designated l-step path of consecutive readers R_i, $p_{T_j} = (R_0, ..., R_i, ..., R_{l-1})$, T_j can pass the step-grained path authentication if it satisfies the following equation:

$$\prod_{i=0}^{l-1} \textsc{Verify}(s_{T_j}^i, d_{R_i}^{\text{aux}}) = 1,$$

where function VERIFY is defined in Equation 1.

A feasible solution should guarantee authentication correctness and security, protect participating entities' privacy, and promise efficiency under practical constraints.

2) ***Correctness****:* The solution should have no false negatives and negligible false positives. First, a false negative occurs when tag T_j faithfully follows its designated path p_{T_j} but fails the authentication, that is,

$$\exists R_i \in p_{T_j} : \ \textsc{Verify}(s_{T_j}^i, d_{R_i}^{\text{aux}}) = 0.$$

Second, a false positive occurs when a tag with invalid state s_{invalid} is accepted by a reader, that is,

$$\exists s_{\text{invalid}} \notin \mathcal{S} \text{ and } \exists R_i \in \mathcal{R} : \ \textsc{Verify}(s_{\text{invalid}}, d_{R_i}^{\text{aux}}) = 1.$$

If R_i happens to be path end, s_{invalid} may make a tag pass path authentication. We consider such false positives as an intrinsic property of cryptography; they are proved negligible by established cryptographic functions. Moreover, such false positives are different from the odds for adversary $\mathcal{A}$ to forge a valid secret. The latter is deemed as a security breach.

3) ***Security****:* A secure solution should guarantee that a probabilistic polynomial time bounded adversary $\mathcal{A}$ cannot forge a valid secret s^{adv} with probability better than random guessing, that is, for any $s^{\text{adv}} \in \mathcal{S}_{\text{possible}}$ chosen by adversary $\mathcal{A}$ and for any $R_i \in p_{T_j}$:

$$Pr(\textsc{Verify}(s^{\text{adv}}, d_{R_i}^{\text{aux}}) = 1) \leq \frac{|\mathcal{S}_{R_i}^{T_j}|}{|\mathcal{S}_{\text{possible}}|} + \epsilon,$$

where $\mathcal{S}_{R_i}^{T_j} \subseteq \mathcal{S}$ represents the set of valid secret(s) for tag T_j on the $(i-1)$th step R_i, $\mathcal{S}_{\text{possible}} \supseteq \mathcal{S}$ represents all possible but not necessarily valid secrets, and ϵ is negligible.

4) ***Privacy****:* Besides preventing security breach, the solution should also avoid privacy leakage. Privacy concerns in the literature fall into two categories, privacy and unlinkability [7]. Privacy protects the exact value of certain secret information while unlinkability thwarts correlating transformed secrets (e.g., via encryption) with tags. Both privacy and unlinkability can be required with granularity of tag, step, or path.

- Identity/Step/Path privacy requires that adversary $\mathcal{A}$ cannot disclose which ID/steps/path a tag corresponds to.
- Tag unlinkability requires that adversary $\mathcal{A}$ cannot correlate secrets with a tag.
- Step/Path unlinkability requires that adversary $\mathcal{A}$ cannot tell whether two tags visit common steps or take the same path.

Blass *et al.* find that 1) for a single tag, tag unlinkability is stronger than identity/step/path privacy, and 2) for two tags, step unlinkability is stronger than path unlinkability [7]. If the solution can satisfy tag unlinkability and step unlinkability, it satisfies other privacy/unlinkability requirements as well.

5) ***Constraints****:* We expect that the solution can efficiently satisfy the preceding requirements under practical constraints.

- Tags perform no computation. Such a complexity constraint makes the solution affordable to widely-deployed low-cost EPC Class 1 Generation 2 tags [28].
- Readers verify path secrets independently (or distributedly). Such an architecture constraint enriches robustness and privacy. First, at each step, not querying a centralized server for secret verification avoids single point of failure. Second, keeping path secrets away from a centralized server further protects supply chain partners' privacy. For example, a partner may not want the centralized server to track its transactions in real time.
- Steps limit minimal path visibility between adjacent readers. Such a policy constraint protects supply chain partners' business strategies [17]. For correct product delivery, a reader should know to which next hop to

transfer a product. Similarly, a reader should also know from which previous hop it receives a product. More specifically, assume that $\overrightarrow{R_{i-1}R_iR_{i+1}}$ is a segment of a designated path. Minimal path visibility requires that R_i know only R_{i-1} and R_{i+1} [17] rather than any other readers on the path (e.g., R_{i-2} or R_{i+2}) [17].

D. Why Not to Adapt Path-Grained Authentication

One may consider step-grained authentication trivially solvable by incremental adaptation of previous path-grained solutions. We acknowledge the feasibility of such an adaption because a designated path represents only one specific instance of valid paths. A feasible adaptation should introduce tag-path bindings that specify which designated path a tagged product should follow. That is, we would grant each step with an additional list of such tag-path bindings. Only tags passing path authentication and belonging to the list are accepted.

This incremental adaptation, however, induces heavy overhead in terms of communication (for the issuer to populate tag-path bindings to steps), storage (for steps to store tag-path bindings), and time (for steps to search over tag-path bindings). Consider, for example, when delivering n tagged products along an l-step designated path. Each step R_i should store the following tag-path binding:

$$R_{i-1}, R_i, R_{i+1} : information\ of\ n\ tags\ from\ R_{i-1}\ to\ R_{i+1}\ via\ R_i.$$

The information of a tag could be its ID together with other related product metadata. Let $|\text{taginfo}|$ denote the size of such information for a tag. Then the storage overhead for each step is $\mathcal{O}(|\text{taginfo}| \times n)$. Distributing such information to all l steps by the issuer introduces a communication overhead of $\mathcal{O}(|\text{taginfo}| \times n \times l)$. Upon path authentication, a step needs to first extract useful metadata from a path secret and then to search whether the medadata is in the issuer-granted list. This search process costs an average time complexity of $\mathcal{O}(\log n)$ and a worst-cast time complexity of $\mathcal{O}(n)$. Considering the huge volume of products in nowadays supply chains, the preceding types of overhead might be heavy and reduce a significant amount of profit.

Furthermore, the preceding storage and communication overhead may be much higher in anonymous RFID systems where tag IDs are not revealed to readers [29], [30]. In such systems, a step/reader R_i cannot simply link tag IDs to a path directive (i.e., R_{i-1}, R_i, R_{i+1}). Instead, it may need to store expected path secrets inside incoming tags. Cryptographic path secrets derived over tag IDs and designated paths are much larger in size than tag IDs. They, therefore, take more communication overhead for the issuer to grant them to readers and take more storage overhead on readers.

StepAuth, on the other hand, does not suffer from the overwhelming overhead for distributing, storing, and searching over tag-path bindings. Besides a comparative communication overhead for initializing reader keys as path-grained authentication, StepAuth imposes only an $\mathcal{O}(1)$ storage overhead and an $\mathcal{O}(1)$ search overhead on each reader, regardless of the number of tags on the designated path and the path length

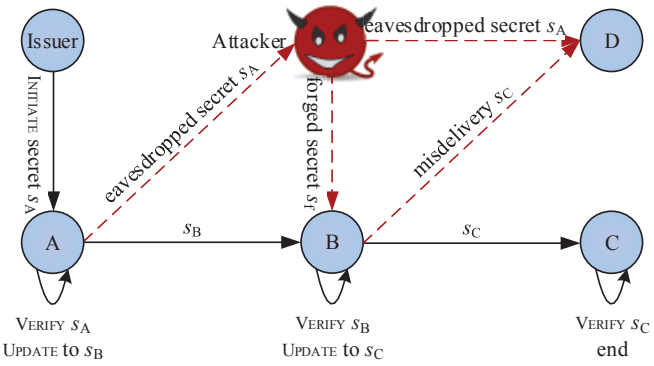

Fig. 1. StepAuth enables each step to locally Verify and Update in-tag path secrets, which also instruct a step which next step to forward the tag. Valid secrets s_A, s_B, and s_C can pass authentication at steps A, B, and C, respectively. Forged (e.g., s_f to step B) or misdelivered (e.g., s_C to step D, or eavesdropped s_A to step D) secrets should fail the authentication.

(Section IV-E for analysis and Section VI-E for evaluation). To the best of our knowledge, StepAuth takes the first step toward a step-grained path authentication solution with all requirements of correctness, security, privacy, and efficiency satisfied.

III. StepAuth Overview

In this section, we outline StepAuth, a step-grained path authentication protocol that satisfies the requirements of correctness, security, privacy, and efficiency defined in Section II-C. StepAuth follows a recursive design. For each step, after receiving a tag, it verifies the path secret inside to make sure that 1) it is the current step on the designated path and 2) which is the next step, updates the path secret such that the next step can verify it in the same way, and finally forwards the tag with updated secret to the next step.

A. Framework

In a nutshell, StepAuth enforces a strict secret-step coupling. That is, at any time, a valid secret would pass the authentication only at its expected step. As shown in Figure 1, valid secrets s_A, s_B, and s_C can pass authentication at steps A, B, C, respectively, rather than any other steps. StepAuth enables each step to locally verify and update path secrets without querying the centralized issuer. Forged or misdelivered secrets should be prohibited from being authenticated. From privacy perspective, StepAuth should also hinder the attacker from correlating secrets s_A, s_B, and s_C.

For step-grained path authentication of tag T_j, StepAuth consists of two key phases (Algorithm 1). In the first phase, issuer $\mathcal{I}$ generates the initial secret $s^0_{T_j}$ based on T_j's designated path $p_{T_j} = (R_0, ..., R_i, ..., R_{l-1})$ and each enroute reader R_i's auxiliary data $d^{\text{aux}}_{R_i}$ (e.g., domain parameters) (line 3). The configuration of $d^{\text{aux}}_{R_i}$ varies per specific secret generation technique. The issuer then stores initial secret $s^0_{T_j}$ to T_j (line 4) and distributes initialized T_j to the first step/reader R_0 on the designated path (line 5). StepAuth now proceeds to the second phase where each reader recursively follows the same procedures. Upon receiving tag T_j, reader R_0 reads its secret $s^0_{T_j}$ (line 8) for verification (line 9). If

Algorithm 1: STEPAUTH: STEP-Grained Path AUTHentication

Input : Tag T_j's designated path $p_{T_j} = (R_0, ..., R_i, ..., R_{l-1})$;
Auxiliary data $d_{R_i}^{\text{aux}}$ of each $R_i \in p_{T_j}$
Output: Path authentication result of tag T_j: $\text{AUTH}(T_j)$

1 $\text{AUTH}(T_j) \leftarrow 0$;
2 //the issuer generates the initial path secret;
3 $s_{T_j}^0 \leftarrow \textbf{INITIALIZE}(p_{T_j}, \forall R_i \in p_{T_j} : d_{R_i}^{\text{aux}})$;
4 $T_j \leftarrow \text{WRITE}(s_{T_j}^0)$;
5 Issue T_j to the first step/reader R_0;
6 //recursive operations by each reader on the designated path;
7 **foreach** $R_i \in p_{T_j}$ **do**
8 $\quad s_{T_j}^i \leftarrow \text{READ}(T_j)$;
9 $\quad$ **if** $\textbf{VERIFY}(s_{T_j}^i, d_{R_i}^{\text{aux}}) = 1$ **then**
10 $\quad\quad R_{i+1}, s_{T_j}^{i+1} \leftarrow \textbf{UPDATE}(s_{T_j}^i, d_{R_i}^{\text{aux}})$;
11 $\quad\quad T_j \leftarrow \text{WRITE}(s_{T_j}^{i+1})$;
12 $\quad\quad$ **if** $i < l-1$ **then**
13 $\quad\quad\quad$ Ship T_j to R_{i+1}, which is the next step;
14 $\quad\quad$ **else**
15 $\quad\quad\quad$ Keep T_j and sell it to a customer at the point of sale;
16 $\quad$ **else**
17 $\quad\quad$ **return** $\text{AUTH}(T_j)$;
18 $\text{AUTH}(T_j) \leftarrow 1$;
19 **return** $\text{AUTH}(T_j)$;

verification succeeds, R_0 updates the secret from $s_{T_j}^0$ to $s_{T_j}^1$ (line 10), writes it to T_j (line 11), and ships it to the next step R_1 (line 13). Otherwise, T_j fails secret verification and thus path authentication (line 17). R_1 and subsequent enroute readers perform the same operations (lines 7-17). For ease of presentation, we introduce the following binary indicator $\text{AUTH}(T_j)$ to demonstrate authentication success with value 1 or failure with value 0.

$$\text{AUTH}(T_j) = \begin{cases} 1, \text{ if } \forall R_i \in p_{T_j} : \text{VERIFY}(s_{T_j}^i, d_{R_i}^{\text{aux}}) = 1; \\ 0, \text{ if } \exists R_i \in p_{T_j} : \text{VERIFY}(s_{T_j}^i, d_{R_i}^{\text{aux}}) = 0. \end{cases}$$

B. Challenges

Among the operations required in Algorithm 1, three are key design challenges—INITIALIZE, VERIFY, and UPDATE. All of them operate on path secrets and are highly correlated. On one hand, how to VERIFY and UPDATE path secrets depends on how INITIALIZE generates path secrets. On the other hand, how to INITIALIZE path secrets should take into account how to simplify VERIFY and UPDATE for imposing less resource demand on readers and tags.

We now dissect construction requirements for path secrets. From functionality perspective, a path secret should satisfy the following two requirements.

- Req 1. For step-grained path authentication, secret $s_{T_j}^i$ can help reader R_i to check whether it is the current step on the designated path.
- Req 2. For product delivery along the designated path, secret $s_{T_j}^i$ can help reader R_i to determine the correct next step R_{i+1}. (Note that a path secret does not need to contain information for verifying previous hop, which can be implied by secret validity.)

From a security and privacy perspective, the path secret should satisfy three other requirements.

- Req 3. For securing the path secret, secret $s_{T_j}^i$ for reader R_i cannot be decrypted by an adversary or other readers without compromising reader R_i.
- Req 4. For securing the path authentication protocol, an adversary or readers cannot forge a valid secret without compromising any encryption keys.
- Req 5. For protecting path privacy, a path secret should keep minimal path visibility between adjacent readers.

Note that Reqs 3-5 are not sufficient for tag/step unlinkability. We will explore sufficient techniques for satisfying them upon presenting STEPAUTH design.

C. Basic Design and Limitations

A basic implementation of STEPAUTH is simply chaining path secrets for each step. Consider a three-step path $\overrightarrow{\text{ABC}}$ for example. To make step A ensure that it is the current step (Req 1) and B is the next step (Req 2), the secret for step A should include A and B. The secret should also be kept secret from adversaries (Req 3) and hard to forge (Req 4). We can accordingly encrypt the secret using a secret key stored on the secure issuer. To limit path visibility, the secret for step A should reveal no more path information other than A and B (Req 5). Following the similar principles for steps B and C, we can construct the path secret for $\overrightarrow{\text{ABC}}$ as the following.

$$\text{Enc}_{k_\text{A}}(\text{A}, \text{B})||\text{Enc}_{k_\text{B}}(\text{B}, \text{C})||\text{Enc}_{k_\text{C}}(\text{C}), \tag{2}$$

where k_A, k_B, and k_C denote the secret keys of A, B, and C, respectively.

As we mentioned, Reqs 3-5 are not sufficient for tag/step unlinkability. The secret by Equation 2 applies to all tags following the same path $\overrightarrow{\text{ABC}}$. This violates both tag unlinkability and step unlinkability if the encryption function $\text{Enc}(\cdot)$ is deterministic.

- First, to guarantee step unlinkability, we should randomize secrets for tags on the same path.

$$\text{Enc}_{k_\text{A}}(\text{A}, \text{B}, \text{Rand}(T_j)||\text{Enc}_{k_\text{B}}(\text{B}, \text{C}, \text{Rand}(T_j))|| \\ \text{Enc}_{k_\text{C}}(\text{C}, \text{Rand}(T_j)).$$

  Since tag IDs are unique across all tags and usually incorporated in path secrets for ease of identification, we use tag T_j's ID as the seed for randomization $\text{Rand}(\cdot)$[4].
- Second, to guarantee tag unlinkability, the secret of a tag should also vary stepwise. We can introduce a pair of encryption and decryption keys for each pair of adjacent readers. Then a step encrypts the secret using the pair-wise key before delivering the tag to the next step. This way, same secrets are differently encrypted on different steps and deter tag correlation.

The basic STEPAUTH design is limited in both efficiency and practicality. First, repeating $\text{Rand}(T_j)$ for step unlinkability exacerbates tag memory cost. Second, it is laborious or even insecure for the issuer to set up pair-wise keys for each pair of neighboring readers. Reader neighborhood is less stable in practical RFID-based supply chains. Partners may

[4] $\text{Rand}(T_j)$ acts as a random initialization vector if $\text{Enc}(\cdot)$ is probabilistic.

become neighbors because of new transactions. Readers may frequently join or leave in dynamic supply chains [15]. One may consider letting two readers independently exchange keys using, for example, Diffie-Hellman. But without a trusted third party as a certificate authority, Diffie-Hellman key exchange is vulnerable to the man-in-the-middle attack [31].

IV. StepAuth Construction

In this section, we construct StepAuth toward practically efficient, distributed, and step-grained authentication of RFID paths. It generates compact path secrets based on a nested encryption technique, which frees StepAuth of repeating random fields $\mathrm{Rand}(T_j)$ and of exchanging keys for neighboring readers. We also leverage hybrid encryption to reap both the security of public key cryptography and the efficiency of symmetric cryptography. Each reader needs to be granted only one decryption key for public key encryption upon the reader's join. Keys for symmetric encryption are included in tag states instead. Encryption keys for public key cryptography to construct tag states are stored on the secure issuer.

A. Nested Encryption

An intuitive way to generate a path secret that satisfies Reqs 1-5 (Section III-B) is nesting a step's secret in that of its previous hop. Consider a three-step path $\overrightarrow{\mathrm{ABC}}$ for example. Using nested encryption, we can initialize A's secret as:

$$\mathrm{Enc}_{k_\mathrm{A}}(\mathrm{A}, \mathrm{B}, \mathrm{Enc}_{k_\mathrm{B}}(\mathrm{B}, \mathrm{C}, \mathrm{Enc}_{k_\mathrm{C}}(\mathrm{C}))).$$

The plaintext first includes identities of A and B. This makes A ensure that it is the current step and B is the next step (Reqs 1 and 2). Using A's key for encryption guarantees that neither an adversary nor other readers can decrypt its secret (Req 3). A's secret should also include the secret for next-hop B to verify.

$$\mathrm{Enc}_{k_\mathrm{B}}(\mathrm{B}, \mathrm{C}, \mathrm{Enc}_{k_\mathrm{C}}(\mathrm{C})).$$

Since it is encrypted using B's key, A cannot decrypt it and thus cannot know which is B's next step if any (Req 5). B performs the same operations on the state as A does. That is, B first verifies that it is the current step and C is the next step; then it forwards the following updated secret to C.

$$\mathrm{Enc}_{k_\mathrm{C}}(\mathrm{C}).$$

After decrypting the secret, C finds only its identifier within and thus knows that it is the path end. Since all secrets are encrypted with readers' encryption keys stored on the issuer, an attacker cannot forge a valid secret without compromising the issuer or any encryption keys (Req 4). Furthermore, secrets vary at each step and thus promise tag unlinkability.

The preceding design, however, should be enhanced toward step unlinkability. If we encrypt only C's identifier into the secret, all tags on the same path will have the same initial secret using deterministic encryption. This violates step unlinkability. In addition, it is difficult to identify the tag at the last step, which is necessary in RFID applications. To address such an issue, we add tag ID into the secret and leverage the uniqueness of tag IDs to randomize secrets for tags following the same path. To avoid any ambiguity, we repeat the identifier of the last step in the initial secret as the following.

$$\mathrm{Enc}_{k_\mathrm{A}}(\mathrm{A}, \mathrm{B}, \mathrm{Enc}_{k_\mathrm{B}}(\mathrm{B}, \mathrm{C}, \mathrm{Enc}_{k_\mathrm{C}}(\mathrm{C}, \mathrm{C}, T_j))). \quad (3)$$

Then if a reader finds itself designated as the "next step" of its own, the reader determines that it is the last step and terminates product delivery.

B. Hybrid Encryption

Generating path secrets using purely public key cryptography (Equation 3) is not practically efficient. First, the maximum length of the message to be encrypted is upper bounded by the size of the encryption key. For example, it is well known that a 1024-bit (128 bytes) RSA key can encrypt a message of length up to 117 bytes following the PKCS#1 v1.5 standard [32]. Since the length of secrets defined in Equation 3 expands with path length, generating long enough keys in favor of long paths would incur heavy computation overhead to the issuer and bandwidth overhead to the issuer-reader channel. Second, public key encryption is much less efficient than symmetric encryption by several orders of magnitude [33]. It induces a heavy overhead for the issuer and readers to process a large amount of tags.

To boost authentication efficiency, we leverage the wisdom of hybrid encryption. Specifically, hybrid encryption uses symmetric cryptography to encrypt/decrypt a message while the symmetric key is encrypted by public key cryptography [34]. Let us apply hybrid encryption to generate the last step C's secret in the preceding example. The message $(\mathrm{C}, \mathrm{C}, T_j)$ now is encrypted using a random symmetric key k^s_C—$\mathrm{Enc}_{k^\mathrm{s}_\mathrm{C}}(\mathrm{C}, \mathrm{C}, T_j)$. To make C capable of decrypting the ciphertext, the issuer should include k^s_C in step C's secret. If the issuer directly include the plaintext of k^s_C, an eavesdropper can also decrypt the secret and breach tag/step privacy. Even worse still, an attacker can forge a secret and breach security. To guarantee security and privacy, the issuer therefore encrypts k^s_C using an encryption key k^e_C. The decryption key k^d_C corresponding to k^e_C is pre-loaded to step C. This way, we can construct step C's secret as the following.

$$secret^\mathrm{C} = \mathrm{Enc}_{k^\mathrm{e}_\mathrm{C}}(k^\mathrm{s}_\mathrm{C}), \mathrm{Enc}_{k^\mathrm{s}_\mathrm{C}}(\mathrm{C}, \mathrm{C}, T_j). \quad (4)$$

Upon receiving the above secret, step C first decrypts $\mathrm{Enc}_{k^\mathrm{e}_\mathrm{C}}(k^\mathrm{s}_\mathrm{C})$ using k^d_C and gets k^s_C. Then it decrypts the remaining part using k^s_C and determines that it is the last step because of double C's. Similarly, B and A's secrets can be constructed as in Equation 5 and Equation 6, respectively.

$$secret^\mathrm{B} = \mathrm{Enc}_{k^\mathrm{e}_\mathrm{B}}(k^\mathrm{s}_\mathrm{B}), \mathrm{Enc}_{k^\mathrm{s}_\mathrm{B}}(\mathrm{B}, \mathrm{C}, secret^\mathrm{C}). \quad (5)$$

$$secret^\mathrm{A} = \mathrm{Enc}_{k^\mathrm{e}_\mathrm{A}}(k^\mathrm{s}_\mathrm{A}), \mathrm{Enc}_{k^\mathrm{s}_\mathrm{A}}(\mathrm{A}, \mathrm{B}, secret^\mathrm{B}). \quad (6)$$

The first step A's secret in Equation 6 is the initial secret to be assigned to the tag. Hybrid encryption promises efficiency for secret generation and verification without sacrificing security and privacy, given that the hybrid encryption is secure against adaptive chosen ciphertext attacks [18].

C. Secret Forgery Attack Resistance

If an attacker can compromise any encryption key k_i^{e} of reader R_i, the attacker can forge a "valid" secret that enables R_i to accept the secret-augmented product and deliver the product to any next hop whose valid secrets have been overheard. The attacker may compromise reader R_i to obtain its decryption key k_i^{d} and domain parameters, with which the encryption key k_i^{e} of R_i might be derived. Take the valid $secret^{\mathrm{B}}$ of B in Equation 5 for example. Consider an attacker that overhears $secret^{\mathrm{B}}$, compromises another reader, say D, and derives its encryption key k_D^{e}. Then the attacker can forge the following secret so as to enable $\overrightarrow{\mathrm{DBC}}$ to be accepted as a designated path.

$$secret^{\mathrm{D}} = \mathrm{Enc}_{k_{\mathrm{D}}^{\mathrm{e}}}(k_{\mathrm{D}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{D}}^{\mathrm{s}}}(\mathrm{D}, \mathrm{B}, secret^{\mathrm{B}}),$$

where $k_{\mathrm{D}}^{\mathrm{s}}$ is a randomly generated symmetric key by D. In an extreme case, an omnipotent attacker may compromise all readers and their encryption keys. It can generate a valid secret for any path it wants to take as if it were the issuer.

To thwart such a secret forgery attack, we further augment a secret with the issuer's signature. The major enhancement is that the issuer uses the same domain parameters to generate a key pair $(k^{\mathrm{pub}}, k^{\mathrm{sec}})$, of which the public key k^{pub} is distributed to each reader upon it joins the supply chain and the secret key k^{sec} is used for the issuer to sign secrets for each step. Upon the signature enhancement, the attacker may generate the following secret to ensure $\overrightarrow{\mathrm{DBC}}$ to be accepted as a designated path.

$$\begin{aligned} secret^{\mathrm{D}+} = {} & \mathrm{Enc}_{k_{\mathrm{D}}^{\mathrm{e}}}(k_{\mathrm{D}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{D}}^{\mathrm{s}}}(\mathrm{D}, \mathrm{B}, secret^{\mathrm{B}+}), \\ & \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{Enc}_{k_{\mathrm{D}}^{\mathrm{e}}}(k_{\mathrm{D}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{D}}^{\mathrm{s}}}(\mathrm{D}, \mathrm{B}, secret^{\mathrm{B}+})), \end{aligned}$$

where we have

$$\begin{aligned} secret^{\mathrm{B}+} = {} & \mathrm{Enc}_{k_{\mathrm{B}}^{\mathrm{e}}}(k_{\mathrm{B}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{B}}^{\mathrm{s}}}(\mathrm{B}, \mathrm{C}, secret^{\mathrm{C}+}), \\ & \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{Enc}_{k_{\mathrm{B}}^{\mathrm{e}}}(k_{\mathrm{B}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{B}}^{\mathrm{s}}}(\mathrm{B}, \mathrm{C}, secret^{\mathrm{C}+})), \end{aligned}$$

and

$$\begin{aligned} secret^{\mathrm{C}+} = {} & \mathrm{Enc}_{k_{\mathrm{C}}^{\mathrm{e}}}(k_{\mathrm{C}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{C}}^{\mathrm{s}}}(\mathrm{C}, \mathrm{C}, T_j), \\ & \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{Enc}_{k_{\mathrm{C}}^{\mathrm{e}}}(k_{\mathrm{C}}^{\mathrm{s}}), \mathrm{Enc}_{k_{\mathrm{C}}^{\mathrm{s}}}(\mathrm{C}, \mathrm{C}, T_j)). \end{aligned}$$

Since the valid secret should be signed by the issuer with its secret key k^{sec}, an attacker cannot forge a valid secret without compromising the issuer. In other words, even if the attacker compromises all readers, it can only verify or update valid secrets instead of forging one.

Based on nested encryption, hybrid encryption, and the preceding unforgeable signature scheme, we next detail how to construct INITIALIZE, VERIFY, and UPDATE.

D. Design: INITIALIZE, VERIFY, and UPDATE

*1) **Configuration**:* The only configuration for STEPAUTH to enable INITIALIZE, VERIFY, and UPDATE is assigning and distributing keys for public key cryptography. Upon a reader joins the supply chain and registers to the issuer, the issuer generates a pair of encryption key and decryption key from system-wide domain parameters for the reader. The issuer keeps the encryption key and securely transmits the decryption key and domain parameters to the reader. An alternative and lighter way is to let each reader generate a key pair using the same domain parameters chosen by the issuer. Then each reader sends its encryption key only to the issuer through a secure channel, and the issuer can use the domain parameters to verify each encryption key following public key cryptography standard. In this case, the issuer does not know readers' decryption keys; it thus promises an even higher security guarantee to prudential supply chain partners. Against the secret forgery attack (Section IV-C), the issuer further generates a pair of public key and secret key. The issuer's public key is also transmitted to each reader. Readers' encryption keys and the issuer's secret key will be used for the issuer to INITIALIZE path secrets, which are then written to corresponding tags. Key distribution and secret initialization are all what the issuer needs to do. The readers themselves will locally VERIFY and UPDATE path secrets carried in tags. Besides the same set of domain parameters, each reader should also be informed of reader-ID length $|R_i|$. That is, domain parameters and $|R_i|$ constitute the auxiliary data $d_{R_i}^{\mathrm{aux}}$. Since $d_{R_i}^{\mathrm{aux}}$ is identical across all readers, we hereafter use d^{aux} for brevity.

Algorithm 2: INITIALIZE by Issuer $\mathcal{I}$

Input : Tag T_j's designated path $p_{T_j} = (R_0, ..., R_i, ..., R_{l-1})$;
Encryption key k_i^{e} and auxiliary data d^{aux} of each $R_i \in p_{T_j}$;
Secret key k^{sec} of the issuer;
Output: Initial secret of tag T_j: $s_{T_j}^0$

$s_{T_j}^0 \leftarrow$ null;
//the issuer generates the initial path secret by calling function H;
$s_{T_j}^0 \leftarrow \mathrm{H}(0)$;
//definition of recursive function H based on an intermediate function F;
$\mathrm{H}(i) = \mathrm{F}(i), \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i))$;
//$\mathrm{Sig}(\cdot)$ is for the issuer to sign the secret;
for $\mathrm{F}(i)$ **do**
  $k_i^{\mathrm{s}} \leftarrow$ a random symmetric key the issuer generates for R_i;
  if $i = l - 1$ **then**
    $\mathrm{F}(i) = \mathrm{Enc}_{k_{l-1}^{\mathrm{e}}}(k_{l-1}^{\mathrm{s}}), \mathrm{Enc}_{k_{l-1}^{\mathrm{s}}}(R_{l-1}, R_{l-1}, T_j)$;
  else
    $\mathrm{F}(i) =$
    $\mathrm{Enc}_{k_i^{\mathrm{e}}}(k_i^{\mathrm{s}}), \mathrm{Enc}_{k_i^{\mathrm{s}}}(R_i, R_{i+1}, \mathrm{F}(i+1), \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i+1)))$;
  //$\mathrm{Enc}(\cdot)$ is performed with auxiliary data d^{aux};
return $s_{T_j}^0$;

*2) **INITIALIZE**:* The issuer generates initial secret $s_{T_j}^0$ by calling a recursive function H (Algorithm 2). Its main design principle is nested encryption, which should be fed with identifiers of tag T_j and each reader R_i on the designated path as well as encryption key k_i^{e} for each reader R_i and the secret key k^{sec} of the issuer. Readers' encryption keys and the issuer's secret key are secure on the issuer. Corresponding decryption key k_i^{d} of reader R_i and public key k^{pub} of the issuer are used for future secret verification by reader R_i. As for symmetric keys k_i^{s}'s for each R_i toward efficient hybrid encryption, the issuer generates them randomly upon initializing secrets. The issuer does not need to remember any of them after their usage.

Although secret verification starts from the first step (Algorithm 1), H composes $s_{T_j}^0$ starting from the last step. For ease of presentation, we introduce another intermediate function F

(lines 7-13), which is called by H as the following (line 5).

$$\mathrm{H}(i) = \mathrm{F}(i), \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i)).$$

The repetition of two R_{l-1}'s (line 10) helps R_{l-1} to determine that it is the last step on the designated path. R_{l-1} thus has no next hop to forward the product. Finally, $\mathrm{H}(0)$ is assigned to initial secret $s^0_{T_j}$ (line 3), which is written to T_j by the issuer.

3) **VERIFY***:* To verify the validity of secret $s^i_{T_j}$ (Algorithm 3), R_i first verifies the issuer's signature $\mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i))$ using the issuer's public key k^{pub} (line 1). Only if the signature verification succeeds can secret verification proceed as follows. R_i first truncates the verified signature $\mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i))$ from $s^i_{T_j}$ (line 2). R_i then decrypts the first field of $s^i_{T_j}$ using its decryption key k^{d}_i to get the symmetric key k^{s}_i (line 3). We then use k^{s}_i to decrypt the second field of $s^i_{T_j}$ (lines 4-5). Let $\mathrm{Plain}(s^i_{T_j})$ denote the corresponding decrypted plaintext. Assume that T_j takes the designated path. If R_i is the last step (i.e., R_{l-1}), $\mathrm{Plain}(s^i_{T_j})$ is (R_{l-1}, R_{l-1}, T_j). Otherwise, $\mathrm{Plain}(s^i_{T_j})$ is $(R_i, R_{i+1}, \mathrm{F}(i+1)), \mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i+1)))$, that is, $(R_i, R_{i+1}, \mathrm{H}(i+1))$. In both cases, R_i matches with the first $|R_i|$ bits of $\mathrm{Plain}(s^i_{T_j})$. Based on this observation, we compare R_i with the leftmost $|R_i|$ bits of $\mathrm{Plain}(s^i_{T_j})$ for verifying $s^i_{T_j}$ (line 5). If a match is found, it means that R_i is the correct current step and secret verification succeeds (line 8). Otherwise, R_i is not supposed to be the current step and secret verification fails (line 10). Verification success and failure respectively return 1 and 0 according to the definition of VERIFY (Equation 1).

4) **UPDATE***:* If secret $s^i_{T_j}$ passes verification, R_i needs to update it to $s^{i+1}_{T_j}$ for the next step R_{i+1} if any (Algorithm 4). R_i first checks whether it is the last step, that is, whether there is a next step R_{i+1}. For the last step R_{l-1}, $\mathrm{Plain}(s^i_{T_j})$ derived in Algorithm 3 (line 5) would be R_{l-1}, R_{l-1}, T_j. It repeats the identifier R_{l-1} twice. Based on this observation, we can determine that R_i is the last step if the leftmost two consecutive $|R_i|$ bits are identical (line 3). Last step R_{l-1} may update the secret as the tag ID T_j (line 5). If R_i has a next step R_{i+1}, it first determines the identifier of R_{i+1} by the second $|R_i|$ bits of (lines 7-8). After removing R_i and R_{i+1} from $\mathrm{Plain}(s^i_{T_j})$, the remaining part $\mathrm{H}(i+1)$ is assigned to $s^{i+1}_{T_j}$ (line 9). R_i now can forward T_j with updated secret $s^{i+1}_{T_j}$ to R_{i+1}. In the same way, R_{i+1} follows Algorithm 3 and Algorithm 4 to verify and update $s^{i+1}_{T_j}$, respectively.

E. Discussions

1) Complexity: Table II provides the complexity of STEPAUTH functions. When the issuer generates the initial secret for an l-step path, INITIALIZE calls a recursive function that casts a step's secret into that of its previous step. For each step, the computation involves a symmetric encryption, a public key encryption, and a signature computation, which account for an $\mathcal{O}(1)$ complexity. INITIALIZE thus takes an $\mathcal{O}(l)$ complexity to generate the initial secret for an l-step path. Upon the secret arrives at each reader, the reader performs signature verification, decryption, and string comparison for VERIFY and string truncation for UPDATE. Both take an $\mathcal{O}(1)$ complexity.

Algorithm 3: VERIFY by Reader R_i

Input : Secret $s^i_{T_j}$;
R_i's decryption key k^{d}_i and auxiliary data d^{aux};
The issuer's public key k^{pub} (stored on each reader);
Output: Verification result: 1 if success, 0 if failure

1 **if** $s^i_{T_j}.\mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i))$ *is verified by* k^{pub} **then**
2 $s^i_{T_j} \leftarrow s^i_{T_j}$ with rightmost $\mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{F}(i))$ removed;
3 $k^{\mathrm{s}}_i \leftarrow \mathrm{Dec}_{k^{\mathrm{d}}_i}(s^i_{T_j}.\mathrm{F}(i).\mathrm{Enc}_{k^{\mathrm{e}}_i}(k^{\mathrm{s}}_i))$;
4 $s^i_{T_j} \leftarrow s^i_{T_j}$ with leftmost $\mathrm{Enc}_{k^{\mathrm{e}}_i}(k^{\mathrm{s}}_i)$ removed;
5 $\mathrm{Plain}(s^i_{T_j}) \leftarrow \mathrm{Dec}_{k^{\mathrm{s}}_i}(s^i_{T_j})$;
6 //$\mathrm{Dec}(\cdot)$ is performed with auxiliary data d^{aux};
7 **if** R_i *matches with leftmost* $|R_i|$ *bits of* $\mathrm{Plain}(s^i_{T_j})$ **then**
8 **return** 1;
9 **else**
10 **return** 0;
11 **else**
12 **return** 0;

Algorithm 4: UPDATE by Reader R_i

Input : $\mathrm{Plain}(s^i_{T_j})$ and reader-ID length $|R_i|$ derived from d^{aux}
Output: Next-step ID R_{i+1} and secret $s^{i+1}_{T_j}$ for R_{i+1} to verify

1 $R_{i+1} \leftarrow$ null;
2 $s^{i+1}_{T_j} \leftarrow$ null;
3 **if** *the leftmost two consecutive* $|R_i|$ *bits of* $\mathrm{Plain}(s^i_{T_j})$ *is identical* **then**
4 R_i is the last step;
5 $s^{i+1}_{T_j} \leftarrow \mathrm{Plain}(s^i_{T_j})$ with leftmost $2|R_i|$ bits removed;
6 **else**
7 $\mathrm{Plain}(s^i_{T_j}) \leftarrow \mathrm{Plain}(s^i_{T_j})$ with leftmost $|R_i|$ bits removed;
8 $R_{i+1} \leftarrow$ leftmost $|R_i|$ bits of $\mathrm{Plain}(s^i_{T_j})$;
9 $s^{i+1}_{T_j} \leftarrow \mathrm{Plain}(s^i_{T_j})$ with leftmost $|R_{i+1}|$ bits removed;
10 **return** R_{i+1} *and* $s^{i+1}_{T_j}$;

2) Key Distribution: STEPAUTH imposes on each reader only its decryption key, the issuer's public key, and corresponding domain parameters. In practical supply chains, it is common that a reader locates on more than one designated paths. For example, a wholesaler may purchase several types of products from different distributors and then sell them to different retailers. For a passing tag, the wholesaler verifies its secret solely according to whether it is specified as the current step, no matter which path the tag is traveling. More specifically, the wholesaler holds a one-to-one mapping with a step instead of one or more paths. Thus, each step of a supply chain requires only one decryption key for it to verify step correctness and update secrets. Since both secret verification and update need no tag computation, STEPAUTH assigns no keys to tags.

3) Scalability and Privacy: The length of an initial secret is linear with path length. Our analysis (Section VI) demonstrates that secrets for practically long paths can be well supported by high-memory EPC Gen2 tags. For example, our implementation of STEPAUTH using ECIES and ECDSA costs 896 bits of secret size per step. A path secret for dozens of steps can perfectly suit for the memory capacity of, for example, Marubeni Chemix and Xerafy 8 KB tags [20] and Fujitsu 64 KB tags [21].

Besides, path secrets leak privacy of path length. Longer secrets indicate longer paths. Given a secret, it is possible to

TABLE II
COMPLEXITY FOR THE ISSUER TO INITIALIZE THE STATE FOR AN l-STEP PATH AND FOR EACH ENROUTE READER TO VERIFY/UPDATE IT.

	Issuer	Reader	
Function	INITIALIZE	VERIFY	UPDATE
Complexity	$\mathcal{O}(l)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

infer how many steps left for the tag to travel. For example, the length of a last-step secret is fixed (line 7 in Algorithm 2). Such a privacy leakage can be easily addressed by adapting STEPAUTH. Since secret update at each step removes a certain number of bits from a secret, we can append equal amount of dummy bits to the secret. To disguise secret-length difference of paths with different lengths, we can predicate a long enough path and use its secret length as a baseline. Secrets of shorter paths can be stretched to the baseline length by appending dummy bits. For simplicity, we omit appending such dummy bits in later discussion. We focus more on the privacy of tag unlinkability and step unlinkability as in the literature.

4) State Extensibility: Current STEPAUTH design encodes only two identifiers of current- and next-step readers for a reader to verify. It is straightforward to extend it to encode additional information in a path secret. For example, a secret may incorporate an expiration time by which a reader should process the product. This is useful in a supply chain of short life-cycle products.

V. SECURITY AND PRIVACY ANALYSIS

In this section, we prove the security and privacy of STEPAUTH. It turns out to evaluate against what attack capabilities STEPAUTH can satisfy security and privacy requirements in Section II-C. Security proof demonstrates that an attacker cannot forge a valid state to pass authentication without compromising the issuer's secret key, even if it can compromise all readers' encryption and decryption keys. Privacy proof demonstrates that an attacker cannot breach tag unlinkability and step unlinkability unless it compromises readers' decryption keys. We do not analyze identity/step/path privacy and path unlinkability as they can be implied by tag unlinkability and step unlinkability, respectively [7].

A. Security

STEPAUTH can satisfy the security requirement if an attacker cannot obtain the issuer's secret key for signing path secrets. That is, the probability for the attacker to forge a valid path secret is no better than random guessing (Section II-C). Actually, it might be impossible for any path authentication protocols to defend against an attacker that have all keys used for generating valid secrets. We thus design STEPAUTH in such a way that critical encryption keys are stored in the more secure issuer. Even if all readers are compromised, an attacker can only verify and update a path secret instead of forging one. This renders our protocols more robust than most of existing path authentication protocols.

Theorem 1. *If the hybrid encryption used in* STEPAUTH *is secure against adaptive chosen ciphertext attacks, a probabilistic polynomial time bounded attacker without compromising the issuer's secret key for signing a valid path p's secret cannot forge a valid secret for path p to pass* STEPAUTH *with probability better than random guessing.*

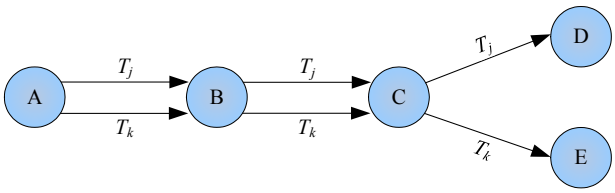

Fig. 2. Paths of two tags T_j and T_k with common steps A, B, and C.

Proof. Because the signature scheme used in STEPAUTH should be secure against adaptive chosen ciphertext attacks, it is non-malleable against adaptive chosen ciphertext attacks [18]. That is, without knowing the issuer's secret key for generating the signature for the secret of path p, a probabilistic polynomial time bounded attacker cannot forge any valid secret for path p with a probability better than random guessing, even if the attacker is provided with any polynomial number of valid secrets selected by the attacker. □

B. Privacy: Tag Unlinkability and Step Unlinkability

STEPAUTH can guarantee both tag unlinkability and step unlinkability against an attacker with only eavesdropping capability (Theorem 2). But both privacy properties will be breached by a stronger attacker that can compromise any readers' decryption keys (Theorem 3).

Theorem 2. *Given that no polynomial-time attacker can break public key cryptography, an attacker with global eavesdropping capability cannot breach tag unlinkability and step unlinkability of* STEPAUTH.

Proof. We sketch the proof using two tags with common steps. As shown in Figure 2, tag T_j takes path $\overrightarrow{\text{ABCD}}$ while tag T_k takes path $\overrightarrow{\text{ABCE}}$. Let $s^{\text{A}}_{T_j}$, $s^{\text{B}}_{T_j}$, $s^{\text{C}}_{T_j}$, and $s^{\text{D}}_{T_j}$ represent the secret of T_j at steps A, B, C, and D, respectively. Similarly, let $s^{\text{A}}_{T_k}$, $s^{\text{B}}_{T_k}$, $s^{\text{C}}_{T_k}$, and $s^{\text{E}}_{T_k}$ respectively denote the secret of T_k at steps A, B, C, and E. We assume that all these eight secrets may be eavesdropped by an attacker.

Tag unlinkability requires that the attacker cannot correlate eavesdropped secrets with tags. Note that for the example in Figure 2, T_j and T_k have same-length paths. They therefore have same-size secrets at steps A, B, and C. Without loss of generality, we discuss that among the four secrets—$s^{\text{B}}_{T_j}$, $s^{\text{C}}_{T_j}$, $s^{\text{B}}_{T_k}$, $s^{\text{C}}_{T_k}$—eavesdropped at steps B and C, whether the attacker can infer which of $s^{\text{C}}_{T_j}$ and $s^{\text{C}}_{T_k}$ is derived from $s^{\text{B}}_{T_j}$. For STEPAUTH, we have

$$\begin{aligned} s^{\text{B}}_{T_j} =& (\text{Enc}_{k^{\text{e}}_{\text{B}}}(k^{\text{s}}_{\text{B}}), \text{Enc}_{k^{\text{s}}_{\text{B}}}(\text{B}, \text{C}, s^{\text{C}}_{T_j}), \\ & \text{Sig}_{k^{\text{sec}}}(\text{Enc}_{k^{\text{e}}_{\text{B}}}(k^{\text{s}}_{\text{B}}), \text{Enc}_{k^{\text{s}}_{\text{B}}}(\text{B}, \text{C}, s^{\text{C}}_{T_j}))). \end{aligned}$$

If the attacker can determine the preceding equality, it can correlate $s^{\text{C}}_{T_j}$ with $s^{\text{B}}_{T_j}$ and breach tag unlinkability. Determining the preceding equality necessitates the knowledge of encryption key k^{e}_{B} of step B, which cannot be inferred by the attacker. STEPAUTH thus provides tag unlinkability.

Step unlinkability requires that the attacker cannot tell whether two tags have more than one common step. Note that it is trivial for the attacker to know that the tags whose

secrets are eavesdropped at the same step have at least one common step. Consider, for example, the four eavesdropped secrets $s^{\mathrm{B}}_{T_j}$, $s^{\mathrm{C}}_{T_j}$, $s^{\mathrm{B}}_{T_k}$, and $s^{\mathrm{C}}_{T_k}$ at steps B and C again. $s^{\mathrm{B}}_{T_j}$ and $s^{\mathrm{B}}_{T_k}$ eavesdropped at step B show that the two tags have a common step B. To further determine their common step C, the attacker needs to correlate $s^{\mathrm{C}}_{T_j}$ eavesdropped at step C with $s^{\mathrm{B}}_{T_j}$ and $s^{\mathrm{C}}_{T_k}$ with $s^{\mathrm{B}}_{T_k}$. This indicates the attacker's ability of breaching tag unlinkability and contradicts with the preceding proof. Therefore, StepAuth provides step unlinkability. □

Theorem 3. *An attacker capable of compromising any readers' decryption keys can breach tag and step unlinkability of* StepAuth.

Proof. We still use the example in Figure 2 to ease discussion. For StepAuth, compromised decryption key $k^{\mathrm{d}}_{\mathrm{B}}$ at step B enables the attacker to correlate $s^{\mathrm{C}}_{T_j}$ with $s^{\mathrm{B}}_{T_j}$ and breach tag unlinkability. Specifically, the attacker uses $k^{\mathrm{d}}_{\mathrm{B}}$ to decrypt the first field of $s^{\mathrm{B}}_{T_j}$ (i.e., $\mathrm{Enc}_{k^{\mathrm{e}}_{\mathrm{B}}}(k^{\mathrm{s}}_{\mathrm{B}})$) and obtains $k^{\mathrm{s}}_{\mathrm{B}}$. Then the attacker uses $k^{\mathrm{s}}_{\mathrm{B}}$ to decrypt the second field of $s^{\mathrm{B}}_{T_j}$ (i.e., $\mathrm{Enc}_{k^{\mathrm{s}}_{\mathrm{B}}}(\mathrm{B},\mathrm{C},s^{\mathrm{C}}_{T_j})$) and obtain $s^{\mathrm{C}}_{T_j}$. Furthermore, the attacker can use $k^{\mathrm{d}}_{\mathrm{B}}$ to decrypt the following $s^{\mathrm{B}}_{T_k}$ as well.

$$\begin{aligned} s^{\mathrm{B}}_{T_k} =&(\mathrm{Enc}_{k^{\mathrm{e}}_{\mathrm{B}}}(k^{\mathrm{s}}_{\mathrm{B}}), \mathrm{Enc}_{k^{\mathrm{s}}_{\mathrm{B}}}(\mathrm{B},\mathrm{C},s^{\mathrm{C}}_{T_k}),\\ &\mathrm{Sig}_{k^{\mathrm{sec}}}(\mathrm{Enc}_{k^{\mathrm{e}}_{\mathrm{B}}}(k^{\mathrm{s}}_{\mathrm{B}}), \mathrm{Enc}_{k^{\mathrm{s}}_{\mathrm{B}}}(\mathrm{B},\mathrm{C},s^{\mathrm{C}}_{T_k}))). \end{aligned}$$

Since both $s^{\mathrm{B}}_{T_j}$ and $s^{\mathrm{B}}_{T_k}$'s plaintexts include steps B and C, the attacker can determine that their corresponding tags have at least two common steps B and C. This breaches step unlinkability. □

VI. Implementation and Performance

In this section, we study an implementation of StepAuth based on the Elliptic Curve Integrated Encryption Scheme (ECIES) [18] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [19]. ECIES is a hybrid encryption scheme proven secure against chosen-plaintext and chosen-ciphertext attackers. We choose ECIES because it is faster with shorter keys yet guarantees the same level of security in comparison with other choices with longer keys. To achieve a 128-bit security, ECIES requires only a 256-bit key while RSA needs a much longer key of 3072 bits [18]. We choose ECDSA as the digital signature scheme as it can use the same set of domain parameters with that of ECIES. Our analysis shows that the StepAuth implementation can generate path secrets hard to forge yet with sizes affordable to high-memory EPC Gen2 tags. It imposes minor computation and storage overhead to the issuer and readers. Comparison results demonstrate that StepAuth is much more efficient than incremental adaptation of path-grained authentication solutions.

A. ECIES Background

We first sketch how ECIES performs encryption and decryption [35]. A set of domain parameters is shared between two communicating entities, say A and B. Among the domain parameters, two critical ones for key generation are generator point G and the order n of G. Without loss of generality, let us consider the case when A transmits an encrypted message to B. We need to generate a pair of keys for B's public key cryptography. The private/decryption key $k_{\mathrm{B}} \in [1, n-1]$ is chosen at random. Then the public/encryption key is derived via scalar point multiplication $K_{\mathrm{B}} = k_{\mathrm{B}}G$. Following the principle of public key cryptography, the encryption key K_{B} is known to A while the decryption key k_{B} is secret on B.

1) ***Encryption****:* To encrypt a message m, A first generates a random number $r \in [1, n-1]$ and calculates $S = rG$ and $P = (P_x, P_y) = rK_B$, where P_x and P_y are affine coordinates of point P. A then feeds P_x and S to a key derivation function (KDF) that is constructed from a hash function and generates two symmetric keys k_{E} and k_{M}. The former is for symmetric encryption of m as $c = \mathrm{Enc}_{k_{\mathrm{E}}}(m)$. The latter is for HMAC of the encrypted message c as $t = \mathrm{HMAC}_{k_{\mathrm{M}}}(c)$ toward CCA security. Finally, the ciphertext A sends to B is $(\overline{S}, c, t)$, where $\overline{S}$ is a compact representation of the elliptic curve point S under point compression [36].

2) ***Decryption****:* Toward correct decryption, B needs also the two symmetric keys k_{E} and k_{M}, which are derived from $P = (P_x, P_y) = rK_B$ with the randomly chosen r unknown to B. B, however, can evolve the equation as $P = (P_x, P_y) = rK_B = rk_{\mathrm{B}}G = k_{\mathrm{B}}rG = k_{\mathrm{B}}S$, of which both factors are known to B as B can recover S from $\overline{S}$. B now can use the same KDF and P_x to derive k_{E} and k_{M}. B first verifies the integrity of the encrypted message by computing $t' = \mathrm{HMAC}_{k_{\mathrm{M}}}(c)$ and check the equality of $t' = t$. If the equality check passes, B further decrypts the encrypted message as $m = \mathrm{Enc}^{-1}_{k_{\mathrm{E}}}(c)$.

B. StepAuth Implementation using ECIES and ECDSA

1) ***Key establishment****:* The issuer chooses a set of domain parameters for ECIES. Upon a reader R_i's registration, the issuer sends the domain parameters to R_i. Again, to more of our interest are the generator point G and the order n of G. R_i first chooses its decryption key k^{d}_i at random from $[1, n-1]$.

$$k^{\mathrm{d}}_i : \in [1, n-1].$$

Then it sets its encryption key k^{e}_i using k^{d}_i and G.

$$k^{\mathrm{e}}_i = k^{\mathrm{d}}_i G.$$

R_i then sends its encryption key k^{e}_i to the issuer, which can verify the encryption key with the domain parameters. Meanwhile, the issuer uses the same set of domain parameters for ECDSA, which generates keys for signing R_i's path secrets. The issuer first chooses its secret key k^{sec} at random from $[1, n-1]$.

$$k^{\mathrm{sec}} : \in [1, n-1].$$

Then the issuer generates its public key k^{pub} using k^{sec} and G as the following.

$$k^{\mathrm{pub}} = k^{\mathrm{sec}} G.$$

The issuer locally keeps the secret key k^{sec} for signing path secrets. It sends the public key k^{pub} to all readers for verifying secret signatures. The issuer and readers communicate through a secure channel.

TABLE III
MEMORY COST BY STEPAUTH USING 256-BIT ELLIPTIC CURVE DOMAIN PARAMETERS SECP256R1 WITH 128-BIT AES AND 128-BIT MD5, GIVEN $|\mathcal{R}|$ 64-BIT INDEXED READERS (I.E., $|R_i| = 64$) AND A 128-BIT INDEXED TAG (I.E., $|T_j| = 128$) STORING SECRET OF AN l-STEP PATH.

Protocol	Issuer	Reader	Tag
STEPAUTH	$\|\mathsf{secp256r1}\| + 256(\|\mathcal{R}\| + 1)$	$< (\|\mathsf{secp256r1}\| + 512 + \log\|R_i\| + \log\|T_j\|)$	$1024 + 896(l-1)$
Notes: $\|\mathsf{secp256r1}\|$ denotes the size of domain parameters. It is around 1546 bits.			

2) **INITIALIZE** *(Algorithm 2):* Besides encryption key k_i^{e} of each enroute reader R_i and secret key k^{sec} of the issuer, the issuer needs also R_i's symmetric key k_i^{s} to generate the initial secret $s_{T_j}^0$. Following the ECIES design, the issuer chooses at random $r_i \in [1, n-1]$ and derives k_i^{s} out of rk_i^{e}. The issuer constructs $s_{T_j}^0$ starting from the last step R_{l-1} of an l-step path as the following.

$$s_{T_j}^{l-1} = \overline{S}_{l-1}, \mathrm{Enc}_{k_i^{\mathrm{s}}}(R_{l-1}, R_{l-1}, T_j), t_{l-1}, \tag{7}$$

where $\overline{S}_i$ is point-compression representation of $S_i = r_i G$ that encrypts symmetric key k_i^{s}. Different from ECIES, t_{l-1} in Equation 7 is no longer an HMAC of $\mathrm{Enc}_{k_i^{\mathrm{s}}}(R_{l-1}, R_{l-1}, T_j)$. Instead, it is the issuer's signature on the first two items using its secret key k^{sec} under ECDSA.

$$t_{l-1} = \mathrm{Sig}_{k^{\mathrm{sec}}}(\overline{S}_{l-1}, \mathrm{Enc}_{k_i^{\mathrm{s}}}(R_{l-1}, R_{l-1}, T_j)).$$

The signature by ECDSA also ensures CCA-security. Following the recursive design of INITIALIZE, we can generate $s_{T_j}^0$ by calling the following function.

$$s_{T_j}^i = \begin{cases} s_{T_j}^{l-1}, & \text{if } i = l-1; \\ \overline{S}_i, \mathrm{Enc}_{k_i^{\mathrm{s}}}(R_i, R_{i+1}, s_{T_j}^{i+1}), t_i, & \text{if } 0 \le i \le l-2, \end{cases} \tag{8}$$

where we have $t_i = \mathrm{Sig}_{k^{\mathrm{sec}}}(\overline{S}_i, \mathrm{Enc}_{k_i^{\mathrm{s}}}(R_i, R_{i+1}, s_{T_j}^{i+1}))$. Note that $s_{T_j}^i$ corresponds to the secret received by R_i.

3) **VERIFY** *(Algorithm 3):* Upon receiving $s_{T_j}^i$, R_i first checks whether it is signed by the issuer. To this end, R_i decrypts t_i and compares the decryption result with the ECDSA-hash digest of $\overline{S}_i$ and $\mathrm{Enc}_{k_i^{\mathrm{s}}}(R_i, R_{i+1}, s_{T_j}^{i+1})$. If the check fails, R_i rejects the tag. Otherwise, R_i removes t_i from $s_{T_j}^i$ and recovers S_i from $\overline{S}_i$ according to domain parameters. Then it derives the symmetric key k_i^{s} using $k_i^{\mathrm{d}} S_i$ and removes $\overline{S}_i$ from $s_{T_j}^i$. R_i further decrypts the remaining $\mathrm{Enc}_{k_i^{\mathrm{s}}}(R_i, R_{i+1}, s_{T_j}^{i+1})$ using k_i^{s} and gets $\mathrm{Plain}(s_{T_j}^i)$. R_i accepts the tag if the first $|R_i|$ bits of $\mathrm{Plain}(s_{T_j}^i)$ match R_i's identifier.

4) **UPDATE** *(Algorithm 4):* Let R_{i+1} represent the second $|R_i|$ bits of $\mathrm{Plain}(s_{T_j}^i)$. If it happens to be $R_{i+1} = R_i$, R_i is the last step. It simply updates the secret by removing two R_i's from $\mathrm{Plain}(s_{T_j}^i)$, which leaves T_j as the final secret. On the other hand, if $R_{i+1} \neq R_i$, R_{i+1} would be the designated next step of R_i. Again, R_i updates the secret by removing the leftmost $2|R_i|$ bits from $\mathrm{Plain}(s_{T_j}^i)$. The remaining content $s_{T_j}^{i+1}$ would be the secret for R_{i+1}, which continues to run VERIFY and UPDATE toward step-grained path authentication.

C. Memory Cost

The memory cost imposed by STEPAUTH on different entities (i.e., the issuer, readers, and tags) depends on the specific configuration of ECIES and ECDSA. Specifically, we choose the 256-bit Elliptic Curve domain parameters secp256r1 that provides a 128-bit security [37]. Both of the issuer's public key and secret key are 256 bits. Both of the encryption and decryption keys for each reader are no longer than 256 bits. The parameter $\overline{S}_i$ is 256 bits. Using 128-bit AES-CBC without initial vector for symmetric cryptography, the derived symmetric key is 128 bits. Using ECDSA for generating secret signature, t_i is 512 bits. Table III summarizes the corresponding memory cost, which is derived next.

The issuer needs to maintain the Elliptic Curve domain parameters, its 256-bit secret key, and encryption keys of all $|\mathcal{R}|$ readers. According to the specification of secp256r1 [37], the domain parameters take around 1546 bits and yield encryption keys up to 256 bits in the compact form under point compression. The memory overhead for the issuer to support STEPAUTH is thus less than $1546+256(|\mathcal{R}|+1)$ bits. Note that the issuer may also maintain the sets of reader/tag identifiers and other supply chain management related information. Such information is intrinsic for supply chain functioning and thus not considered as overhead imposed by our STEPAUTH. In practice, we suggest to choose the size of reader ID $|R_i| = 64$ bits and the size of tag ID $|T_j| = 128$ bits. In such case, we have $\log|R_i| = 6$ and $\log|T_i| = 7$.

Each reader needs to keep auxiliary data mainly including the domain parameters, a decryption key, and the issuer's 256-bit public key. Using the secp256r1 domain parameters, a decryption key is no longer than 256 bits. Besides, the auxiliary data includes also the size of reader identifier R_i and of tag identifier T_j. A reader requires allocating additional $\log|R_i|$ bits and $\log|T_j|$ bits to store reader-ID size and tag-ID size, respectively, a reader thus takes less than $|\mathsf{secp256r1}| + 512 + \log|R_i| + \log|T_j|$ bits to support STEPAUTH.

Tag memory cost is our major concern. Each tag needs to store a path secret generated by STEPAUTH using the path the tag should take. If secrets for long paths way exceed the storage capacity of tags, STEPAUTH would be impractical. Per STEPAUTH design, the initial secret of a given l-step path is the longest among all secrets corresponding to enroute readers. Since STEPAUTH constructs the initial secret from the last step. We now analyze the size of the initial secret from backward. The secret for the last step is $s_{T_j}^{l-1} = (S_{l-1}, \mathrm{Enc}_{k_i^{\mathrm{s}}}(R_{l-1}, R_{l-1}, T_j), t_{l-1})$. Note that we use AES-CBC mode, which guarantees security without initial vector when used with a random encryption key. Given 64-bit reader IDs and 128-bit tag IDs, we derive the size of $s_{T_j}^{l-1}$ as:

$$|s_{T_j}^{l-1}| = 256 + (64 + 64 + 128) + 512 = 1024 \text{ (bits)}.$$

For one previous step after another, the secret expands with one S_i, two reader IDs R_i and R_{i+1}, and one t_i (Equation 8), which account for $256 + 64 + 64 + 512 = 896$ bits. Then the size of R_i's secret is derived as the following.

$$|s_{T_j}^i| = 1024 + 896((l-1) - i) \text{ (bits)}.$$

TABLE IV
COMPUTATION COST BY STEPAUTH USING SECP256R1 WITH 128-BIT AES AND 128-BIT MD5. THE METRICS ARE THE TIME FOR THE ISSUER TO GENERATE THE INITIAL SECRET FOR AN l-STEP PATH AND FOR EACH ENROUTE READER TO VERIFY THE SECRET.

Protocol	Function	Cryptographic Operation	Computation Time using Different Benchmarks (ms)	
			Windows [22]	Linux [23]
STEPAUTH	Issuer.INITIALIZE	(ECIES Encryption + ECDSA Signature) $\times\, l$	$(5.65 + 2.88) \times l = 8.53l$	$(2.58 + 1.31) \times l = 3.89l$
	Reader.VERIFY	ECDSA Verification + ECIES Decryption	$2.88 + 8.53 = 11.41$	$4.07 + 1.75 = 5.82$

We, therefore, derive the size of the initial secret $s^0_{T_j}$ as

$$|s^0_{T_j}| = 1024 + 896(l-1) \text{ (bits)}.$$

Such secret lengths can be practically supported by available high memory EPC Gen2 tags such as 8 KB tags by Marubeni Chemix and Xerafy [20] and 64 KB tags by Fujitsu [21]. Even for a practically long enough path of 100 steps, the initial secret generated by STEPAUTH is up to $1024 + 896 \times (100 - 1) = 89728$ bits $= 10.95$ KB, which perfectly suits for the capacity of high memory tags.

D. Computation Cost

We estimate the computation cost brought by STEPAUTH using the Crypto++ 5.6.0 Benchmarks under both Windows (Vista 32-bit with Intel Core 2 1.83 GHz) [22] and Linux (with AMD Opteron 8354 2.2 GHz) [23] environments. Rather than UPDATE that involves simple string operations, we focus more on INITIALIZE and VERIFY that enforce complex cryptographic computation. More specifically, we measure the computation cost by the time for the issuer to generate the initial secret for an l-step path and the time for an enroute reader to verify its corresponding secret. To incorporate a step into the secret, the issuer performs two cryptographic operations, one for ECIES encryption and one for ECDSA signature. It thus takes the issuer (ECIES Encryption + ECDSA Signature) $\times\, l$ operations to generate the initial secret for an l-step path. When a reader verifies a secret, it processes data related to only one step. The reader first performs signature verification and then ECIES decryption. Table IV summarizes STEPAUTH's computation cost. It takes less than 1 s to generate the initial secret for a 100-step path and around 10 ms to verify a secret.

E. Comparison with Incremental Adaptation of Path-Grained Authentication

Finally, we quantify the performance gap between the baseline solution based on incremental adaptation of path-grained authentication (Section II-D) and STEPAUTH. Among the existing path-grained authentication protocols listed in Table I, we choose CHECKER [13] to adapt because its path secret supports distributed verification. A secret generated by CHECKER has a constant size of 960 bits; it is verified and updated step wise. To support a step-grained authentication of a designated path, the adaptation of CHECKER should pre-assign a tag's step-wise secret to each corresponding step along the designated path (Section II-D). Both the overall communication overhead for the issuer to transmit these secrets to readers and the overall storage overhead on readers can be approximated as the following.

$$n \times l \times 960 \text{ bits},$$

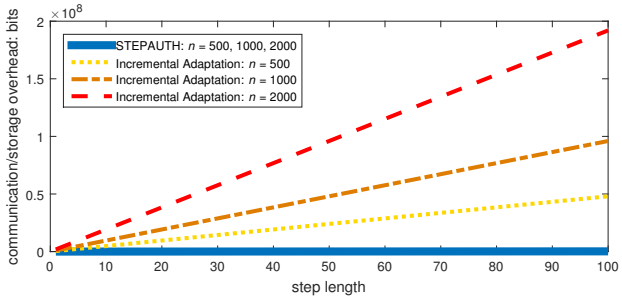

Fig. 3. Comparison of communication/storage overhead on readers between STEPAUTH and incremental adaptation of CHECKER. STEPAUTH uses 256-bit Elliptic Curve domain parameters secp256r1 with 128-bit AES and 128-bit MD5, given $|\mathcal{R}|$ 64-bit indexed readers (i.e., $|R_i| = 64$) and a 128-bit indexed tag (i.e., $|T_j| = 128$). CHECKER uses 960-bit secrets.

where we have n tagged products to be delivered along paths with an average number l of steps. The communication overhead affects how much time the issuer takes to prepare all readers with necessary information for secret verification. That is, it affects the initialization time. Moreover, the storage overhead on readers affects how much time a reader takes to verify a secret. The more secrets a reader stores, the longer time it will take to verify whether a secret is locally stored. On the other hand, the above overall communication/storage overhead for our STEPAUTH is as the following (Section VI-C).

$$l \times (|\text{secp256r1}| + 512 + \log|R_i| + \log|T_j|) \text{ bits}.$$

It is regardless of the number n of tags as STEPAUTH stores secrets in tags instead of readers.

We compare the overhead of STEPAUTH and the incremental adaptation of CHECKER in Figure 3. The efficiency superiority of STEPAUTH over the incremental adaptation intensifies as path length and tag number increase.

VII. LIMITATIONS

In this section, we discuss limitations of STEPAUTH and provide countermeasures. As our discussions will show, some limitations affect all path authentication protocols.

A. Hardness of Constant-size Secret

We find it hard for a step-grained authentication solution to construct constant-size path secrets regardless of the path length. An ideal constant-size path secret has a fixed size, whether the corresponding path is as short as one step or as long as, say, 100 steps. This way, we not only save tag memory but also protect path privacy in terms of its length. Simply extending all secrets to the length of the longest path's secret by padding dummy bits (Section IV-E) preserves path-length privacy but consumes tag memory.

We now investigate the impossibility of constant-size path secrets using the information theory. We deduce the conclusion

by contradiction whereby we first assume that a constant-size path secret regardless of path length exists. For ease of discussion, we consider the entropy of the constant-size path secrets smaller than a large enough constant C. Take an n-step path $P_n = (R_0, R_1, ..., R_i, ..., R_{n-1})$ for example. The initial secret for path P_n should contain sufficient information for each enroute reader R_i to derive its corresponding path segment (i.e., (R_i, R_{i+1}) for $0 \leq i < n-1$ and (R_{n-1}, R_{n-1}) for $i = n-1$). Let $H(P_n)$ and $H(R_i)$ represent the entropies of path P_n's initial secret and each reader R_i's corresponding path secret, respectively. By the Source Coding Theorem [38], we have the following equation.

$$H(P_n) \geq \sum_{i=0}^{n-1} H(R_i) \geq nd,$$

where $0 < d \leq \min\{H(R_i) \mid 0 \leq i \leq n-1\}$. Given the assumption that a constant-size path secret exists and its entropy is less than a constant C, we have $C > nd$ for path P_n with any length of n. This is contradictory if n is large enough. Therefore, it is unreasonable to assume that a constant-size path secret regardless of path length. In other words, we have proven the impossibility of constant-size secret construction for step-grained path authentication.

B. Impact of Replay/Clone Attack

An attacker may replay eavesdropped valid secrets. Such a replay attack resonates with clone attacks, which flash compromised data of a genuine tag to a clone tag. Then the clone one can behave exactly the same as its genuine counterpart during authentication. We consider the clone attack as a concern for not only StepAuth but also any other path authentication solutions. Most clone attacks deploy clone tags at different places than that of their genuine counterparts [39], [40]. We find that StepAuth can intrinsically filter such clone tags. StepAuth enforces step-grained path authentication. That is, a secret is deemed valid only at its expected step. Take state $s_{T_j}^{\mathrm{B}}$ of T_j at step B in Figure 2 for example. It will fail authentication at any other steps rather than step B.

A sophisticated attacker may replay a clone tag with the valid secret at where it is eavesdropped. There are protocols to detect clone tags colocating with their genuine counterparts [30]. However, a smarter attacker would delay replaying/deploying a clone tag, say, with state $s_{T_j}^{\mathrm{B}}$ at step B after T_j is shipped away. To detect such clone tags, each step should maintain a list of authenticated secrets. Then before authenticating each upcoming tag, the reader first checks whether its secret is already in the list. If yes, the tag is suspicious and discarded. We suggest how to reduce time and storage overhead below and leave design details for future work. To reduce time overhead, a Bloom filter can be introduced to track secrets in the list [41]. A Bloom filter helps to quickly verify that a secret is not in the list. Since a secret not in the list can also be regarded as in the list with a (controllable) slight probability, we need to search over the list for in-secrets to avoid mis-regarding genuine tags as clones. To reduce storage overhead, we can limit the effectiveness of tag secrets within a certain time duration. Specifically, we divide the time to epochs and assign each epoch with a unique index. We then enclose each tag secret with an index of the corresponding time epoch. A step needs to store authenticated secrets within only recent epochs.

C. Tolerance of Alternative Paths

It is possible that the only designated path for a product encounters step failures. When a step failure occurs, we need alternative paths to deliver the affected product. Alternative paths come into play upon two cases. The first case is concerned with dynamic supply chains where new readers may join and existing readers may leave during products are travelling across the supply chain. In this case, paths are hard to predict and therefore challenging to pre-generate path secrets. To our knowledge, there is only one existing work by Cai *et al.* [15] to study path authentication in dynamic RFID-enabled supply chains. The solution therein requires each reader to query the centralized issuer for online secret verification. All the other path authentication solutions concentrate on static supply chains where all valid paths are fixed. The second case for alternative paths is therefore in such static supply chains. Since we generate very compact path secret (i.e., 896 bits per step), it is practical for high-memory tags with dozens of kilobytes to accommodate hundreds of steps' secrets. We, therefore, suggest generating a secret for each of the alternative paths and store all the secrets in the tag. Then no matter which path the tag follows, step-grained authentication can be performed.

We would like to further emphasize that we do not propose step-grained authentication to replace existing path-grained authentication. Instead, we consider the proposed solution much more practically efficient for scenarios where products do follow designated paths. Typical examples are build-to-order supply chains and express delivery services for expensive products. In such scenarios, intermediate readers/steps should be highly reliable. In other words, even if alternative paths need to be considered, the number of them should be limited. We, therefore, consider the size of multiple paths' secrets affordable to tag memory; at least we can attach multiple tags to a product when necessary.

D. Vagueness of Tag-Product Binding

Another concern of StepAuth is that incorrect tag-product binding may make a product delivered toward an unexpected destination. Consider for example two products A and B with designated paths of (R_0, R_1, R_2) and (R_0, R_3, R_4), respectively. We first generate two path secrets. The secret for product A is stored in tag T_{A} and the other secret in tag T_{B}. If we attach tag T_{A} to product B by mistake, then product B will be delivered along A's designated path (R_0, R_1, R_2), instead of the expected (R_0, R_3, R_4).

We consider the preceding concern as an intrinsic limitation of RFID-based supply chain management. It affects not only any path authentication solution but also other management operations. The key reason is that in an RFID-enabled supply chain, tag genuineness is regarded as product authenticity. It is said that in supermarkets or shopping malls, a customer

could replace tags on expensive products with those on cheap ones. This way, the customer can buy expensive products at lower prices. To put it another way, the information carried in tags serves as the credential for authenticating products. Generalizing this to a larger context of authentication, a valid credential makes any entity pass authentication if the credential is the solely required factor.

One feasible countermeasure is to attach fragile RFID tags to products [42], [43]. If an attacker peels off a fragile RFID tag from the attached product, the tag IC would be completely destroyed (i.e., break on removal). It is thus impossible for tag replacement to work. Another feasible countermeasure needs to introduce additional factors for authentication. Back to the RFID-enabled supply chain management, product specifics (e.g., appearance) could be leveraged to strengthen RFID-based authentication. Take the preceding tag replacement feasibility for instance again. If the customer sticks a tag removed from a hundreds-of-dollars TV screen to another more expensive thousands-of-dollars one, a cautious cashier would find it suspicious when the customer checks out and the price prompt on the computer is unreasonably low.

E. Reliance on Secure Channel

As with most RFID path authentication protocols, StepAuth assumes a secure channel for issuer-reader communication. Certainly, it is more practical to remove this assumption. While existing solutions do not wrestle with an insecure channel, the focus is how to generate secure and efficient secrets for path authentication given established keys.

We suggest a lightweight enhancement of the key establishment process for StepAuth. The enhancement imposes only one more round of communication. The motivation is to leverage public key cryptography, which is designed for secure communication over a potentially insecure channel. Since StepAuth requires the issuer to first transmit its public key k^{pub} and domain parameters to a reader, the goal of the enhancement is to protect the secrecy of k^{pub} and the domain parameters. Later messages from the reader to the issuer can be encrypted using k^{pub} and only the issuer can decrypt them. To protect the initial issuer-to-reader message, the reader generates a pair of public key $k_{\mathrm{R}}^{\mathrm{pub}}$ and secret key $k_{\mathrm{R}}^{\mathrm{sec}}$ under any proven secure public cryptography scheme. The reader manages to obtain a public key certificate for its public key $k_{\mathrm{R}}^{\mathrm{pub}}$ and transmits it to the issuer. The issuer verifies the certificate and if it is correct, uses $k_{\mathrm{R}}^{\mathrm{pub}}$ to encrypt its message to the reader, which can decrypt the message using $k_{\mathrm{R}}^{\mathrm{sec}}$. After establishing keys for ECIES and ECDSA, the reader no longer needs to store $k_{\mathrm{R}}^{\mathrm{pub}}$ and $k_{\mathrm{R}}^{\mathrm{sec}}$. We turn to ECIES and ECDSA after then because of their high efficiency.

VIII. Conclusion

We have proposed the first step-grained RFID path authentication protocol called StepAuth toward combating counterfeit products in supply chains. Different from prior path authentication protocols, StepAuth enforces a practical deliver regulation that requires products be delivered along designated paths. Such a regulation is critical for scenarios like build-to-order supply chain management and express delivery service where products are delivered to specific receivers. StepAuth encodes the path into an in-tag secret, which enables each reader to locally authenticate tag validity and make delivery decision. What makes StepAuth shine more is that it limits minimal path visibility disclosure between only adjacent readers. This protects supply chain partners' privacy such as transaction strategy. Most traditional path authentication protocols, however, leak more path visibility as they usually offload path sets to readers. To make StepAuth practically efficient, we leverage nested encryption and hybrid encryption to quickly generate and verify path secrets. The generated secrets naturally suit for the storage capacity of available high-memory EPC Gen2 tags. StepAuth guarantees tag unlinkability and step unlinkability against attackers that cannot compromise decryption keys on readers. Furthermore, we prove that StepAuth can prohibit an attacker from forging valid secrets even if it can compromise all readers.

Acknowledgment

This work is supported in part by the National Science Foundation of China under Grant No. 61402404. We also thank Editors and Reviewers of IEEE Transactions on Information Forensics and Security, Yutian Yang, and Avery Laird for their review efforts and helpful feedback.

References

[1] K. Ouafi and S. Vaudenay, "Pathchecker: An RFID application for tracing products in supply-chains," in *RFIDSec*, 2009.

[2] Global trade in fake goods worth nearly half a trillion dollars a year - OECD & EUIPO. [Online]. Available: https://www.oecd.org/industry/global-trade-in-fake-goods-worth-nearly-half-a-trillion-dollars-a-year.htm

[3] In China, Fear of Fake Eggs and 'Recycled' Buns. [Online]. Available: http://www.nytimes.com/2011/05/08/world/asia/08food.html

[4] Counterfeit watches and jewelry are the new counterfeit handbags. [Online]. Available: https://qz.com/376249/counterfeit-watches-and-jewelry-are-the-new-counterfeit-handbags/

[5] Officials Tell Fans How to Spot Fake Super Bowl Tickets. [Online]. Available: http://www.nbcdfw.com/news/sports/Officials-Tell-Fans-How-to-Spot-Fake-Super-Bowl-Tickets-412602683.html

[6] Counterfeit electronics can be 'highly dangerous'. [Online]. Available: http://www.cbc.ca/news/canada/toronto/counterfeit-electronics-can-be-highly-dangerous-1.2744581

[7] E.-O. Blass, K. Elkhiyaoui, R. Molva, and E. S. Antipolis, "Tracker: security and privacy for RFID-based supply chains," in *NDSS*, 2011.

[8] S. Cai, R. H. Deng, Y. Li, and Y. Zhao, "A new framework for privacy of RFID path authentication," in *ACNS*, 2012, pp. 473–488.

[9] H. Wang, Y. Li, Z. Zhang, and Y. Zhao, "Efficient tag path authentication protocol with less tag memory," in *ISPEC*, 2016, pp. 255–270.

[10] M. S. I. Mamun and A. Miyaji, "RFID path authentication, revisited," in *IEEE AINA*, 2014, pp. 245–252.

[11] B. Ray, M. Chowdhury, and J. Abawaiy, "Puf-based secure checker protocol for networked RFID systems," in *IEEE ICOS*, 2014, pp. 78–83.

[12] H. Wang, Y. Li, Z. Zhang, and Z. Cao, "Two-level path authentication in epcglobal network," in *IEEE RFID*, 2012, pp. 24–31.

[13] K. Elkhiyaoui, E.-O. Blass, and R. Molva, "Checker: On-site checking in RFID-based supply chains," in *ACM WiSec*, 2012, pp. 173–184.

[14] A. Gunasekaran and E. W. Ngai, "Build-to-order supply chain management: a literature review and framework for development," *Journal of operations management*, vol. 23, no. 5, pp. 423–451, 2005.

[15] S. Cai, Y. Li, and Y. Zhao, "Distributed path authentication for dynamic RFID-enabled supply chains," in *IFIP International Information Security Conference*, 2012, pp. 501–512.

[16] (2015) How Many Products Does Amazon Sell? [Online]. Available: https://export-x.com/2015/12/11/how-many-products-does-amazon-sell-2015/

[17] T. Burbridge and A. Soppera, "Supply chain control using a RFID proxy re-signature scheme," in *IEEE RFID*, 2010, pp. 29–36.
[18] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.
[19] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.
[20] A Flurry of High-Memory Tags Take Flight. [Online]. Available: http://www.rfidjournal.com/articles/view?8295
[21] Fujitsu Develops World's First 64KByte High-Capacity FRAM RFID Tag for Aviation Applications. [Online]. Available: http://www.fujitsu.com/global/about/resources/news/press-releases/2008/0109-01.html
[22] Crypto++ 5.6.0 Benchmarks (Windows). [Online]. Available: https://www.cryptopp.com/benchmarks.html
[23] Crypto++ 5.6.0 Benchmarks (Unix). [Online]. Available: https://www.cryptopp.com/benchmarks-amd64.html
[24] R. Angeles, "RFID technologies: supply-chain applications and implementation issues," *Information systems management*, 2005.
[25] S. Sarma, "Introductory Talk: Some issues related to RFID and security," in *Workshop on RFID Security*, 2006.
[26] R.-I. Paise and S. Vaudenay, "Mutual authentication in RFID: security and privacy," in *ACM AsiaCCS*, 2008, pp. 292–299.
[27] R. Koh, E. W. Schuster, I. Chackrabarti, and A. Bellman, "Securing the pharmaceutical supply chain," *White Paper, Auto-ID Labs, Massachusetts Institute of Technology*, pp. 1–19, 2003.
[28] Class-1 Generation-2 UHF RFID. [Online]. Available: www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf
[29] M. Kodialam, T. Nandagopal, and W. Lau, "Anonymous tracking using RFID tags," in *IEEE INFOCOM*, 2007, pp. 1217–1225.
[30] K. Bu, X. Liu, J. Luo, B. Xiao, and G. Wei, "Unreconciled collisions uncover cloning attacks in anonymous RFID systems," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 423–439, 2013.
[31] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta *et al.*, "Imperfect forward secrecy: How diffie-hellman fails in practice," in *ACM CCS*, 2015, pp. 5–17.
[32] RFC 2313 - PKCS #1: RSA Encryption Version 1.5 - IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc2313
[33] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2011.
[34] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM Journal on Computing*, vol. 33, no. 1, pp. 167–226, 2003.
[35] Integrated Encryption Scheme. [Online]. Available: https://en.wikipedia.org/wiki/Integrated_Encryption_Scheme
[36] A. Jivsov, "Compact representation of an elliptic curve point," 2014.
[37] M. Qu, "Sec 2: Recommended elliptic curve domain parameters," 1999.
[38] D. MacKay, *Information theory, inference, and learning algorithms*, 1st ed. Cambridge University Press, 2003.
[39] D. Zanetti, L. Fellmann, and S. Capkun, "Privacy-preserving clone detection for RFID-enabled supply chains," in *IEEE RFID*, 2010.
[40] K. Bu, M. Weng, Y. Zheng, B. Xiao, and X. Liu, "You can clone but you can't hide: A survey of clone prevention and detection for rfid," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1682–1700, 2017.
[41] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[42] The World's Best Anti-counterfeit RFID Label Supplier. Fragile label, Easy Shredding. [Online]. Available: https://tinyurl.com/y7b5gutd
[43] Disposable Printed Fragile RFID Tag Label For Loyalty System, 3M Adhesive Layer. [Online]. Available: https://tinyurl.com/y9dpu7rx

Kai Bu received the B.Sc. and M.Sc. degrees in computer science from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2006 and 2009, respectively, and the Ph.D. degree in computer science from The Hong Kong Polytechnic University, Kowloon, Hong Kong, in 2013. Currently, he is an Assistant Professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include networking and security. He is a Member of the ACM, the IEEE, and CCF. He is a recipient of the Best Paper Award of IEEE/IFIP EUC 2011 and the Best Paper Nominees of IEEE ICDCS 2016.

Yingjiu Li is currently an Associate Professor in the School of Information Systems at Singapore Management University (SMU). His research interests include RFID Security and Privacy, Mobile and System Security, Applied Cryptography and Cloud Security, and Data Application Security and Privacy. He has published over 130 technical papers in international conferences and journals, and served in the program committees for over 80 international conferences and workshops. Yingjiu Li is a senior member of the ACM and a member of the IEEE Computer Society. The URL for his web page is http://www.mysmu.edu/faculty/yjli/.