Two Rounds RFID Grouping-Proof Protocol

Sarah Abughazalah, Konstantinos Markantonakis, Keith Mayes
Smart Card Centre-Information Security Group (SCC-ISG)
Royal Holloway, University of London
Email: {Sarah.AbuGhazalah.2012, K.Markantonakis, Keith.Mayes}@rhul.ac.uk

***Abstract*—In this paper, we focus on a particular RFID application called a grouping-proof, where an entity such as an RFID reader generates a proof that two or more tags have been scanned simultaneously. This proof is then verified by the server, which processes the tags' data in the system. However, designing a grouping-proof protocol is challenging for two reasons. Firstly, in some cases, the server that authenticates the tags is offline during the scanning process, thus the tags' data need to be assembled in a valid grouping-proof to be authenticated later. Secondly, a number of recent grouping-proof protocols are either prone to attacks, or they are not efficient in terms of performance. In this paper, we present an offline two rounds grouping-proof protocol that provides immunity against well-known attacks on RFID protocols and improves tag's memory and computing performance. We analysed our protocol using a mechanical formal analysis tool called Scyther, which did not find any feasible attack(s). Finally, an implementation of the proposed protocol is conducted to measure the tag's memory space and computing time cost using an IAIK UHF RFID tag emulator.**

***Index Terms*—RFID, Secuty, Privacy, Scyther, Grouping-proof, Protocol**

I. Introduction

Radio Frequency Identification (RFID) is a wireless technology that uses radio signals to identify tags attached to objects [1]. Ultar-High Frequency (UHF) passive tags are wireless transponders that do not have any power of their own and only respond to the electromagnetic fields generated by nearby reader(s), its frequency band ranges from 860 MHz to 950 MHz. The reader communicates with the server that manages and processes tags' data [1].

One of the RFID standards that is widely-used and associated with passive RFID tags is the ISO18000-6C mostly known as EPCglobal UHF Class-1 Generation-2 (hereinafter denoted as EPC Gen2). This standard is used with applications that require long distance communication up to 10 meters such as in supply chain for tracking purposes [2]. The RFID tools used in this paper support this standard.

One of the important features of RFID technology is its ability to generate a proof that two or more legitimate tags have been scanned simultaneously by an RFID reader within its broadcast range. The first proof was introduced by Juels [3]. It involved only two tags and was called the "yoking-proof". Since its introduction, the yoking-proof has evolved to include multiple tags [4] and is now known as the "grouping-proof". A grouping-proof can be used in many systems including [5]:

- Hospitals: proving that a certain patient has been given his/her medications at the same time.
- Manufacturing: proving that devices have been sold with their attachments.
- Access control: establishing that a group of people with legitimate RFID token were present.
- Supply chain: proving that tagged products were shipped together in a group.

In such systems, the server, which can be an auditing-body or a verifier, might not participate in the scanning process, but at a later time, it verifies the legitimacy and

existence of the tags.

In a typical grouping-proof scenario where there is *n* RFID tags in the group [6], the i^{th} tag (T_i), i.e. $1 \leq i \leq n$, sends a message (M_i) to the reader, then the reader transfers M_i to the next tag (T_{i+1}) in the same group and waits for its response. The reader repeats this operation *n* times. Finally, when the reader receives *n* responses, it creates the proof and sends it to the server. Hence, the number of rounds is proportional to the number of tags in the group [6]. In this paper, we minimised the number of rounds from *n* rounds to *two* rounds.

II. Motivation

There are two modes of verifying the authenticity of RFID tags; online and offline. In the online mode, the server that verifies the proof is running during the protocol execution, while in the offline mode the server is not present during the scanning process. Where the server is online, the solution is straightforward as each tag can authenticate itself directly to the server. On the other hand, if the server is offline the solution is challenging from the security side as the generated proof is verified later [7].

In this paper, we aim to propose a grouping-proof protocol that is provably secure against well-known attacks on RFID systems, and to improve tag's performance by providing the following features:

- Minimise the number of rounds to reduce time delay.
- Concurrency, where each tag does not need to wait for the T_{i-1} message to respond. Hence, dependency between tags is omitted.
- Reading order independent, where the tags can be verified by the server in any order. This approach reduces failure rates [8].

Many researchers have proposed RFID yoking/grouping-proof protocols over the last decade [3]–[16]. However, most of them are shown to be insecure as illustrated in [12], [17], and/or do not provide forward secrecy as shown in [5]. In addition, the proposed RFID grouping-proof protocols are not efficient from a performance perspective as they require the tags to participate in *n* rounds, the tags' responses depend on the predecessor tags, and require large memory and computing costs such as in [4], [5], [7], [13]–[15]

Moriyama et al. [16] proposed two rounds RFID grouping-proof protocol, however their protocol is vulnerable to reader impersonation attack and did not provide forward secrecy. Hence, developing a secure and efficient RFID grouping-proof protocol is still challenging and ongoing. A comparison between recent grouping-proof protocols and our protocol is shown in Section V.

A. Contributions

The main contributions of this study are:

1) Our proposed protocol is the only two rounds grouping-proof protocol that is provably secure against well known attacks on RFID tags namely, replay attacks, impersonation attacks, desynchronisation incidents, location tracking and forward secrecy invasion.
2) To the best of our knowledge, RFID grouping-proof in previous attempts did not combine the features stated in the previous section to generate an efficient RFID grouping-proof.
3) The proposed protocol is provably secure against passive RFID well-known attacks.
4) We implemented our proposed protocol to measure the tag's operating performance, and we found that our protocol consumes low memory space and low computing time cost.

B. Structure of the paper

The rest of this paper is organised as follows: In Section III, the protocol's main goals are discussed. In Section IV, the protocol is explained in detail. In Section V, the protocol is informally and formally

analysed. Section VI presents the experimental work and performance measurement. In Section VII, we offer concluding remarks.

III. Protocol Goals

The proposed protocol should meet the following goals:

1) Forward secrecy: The proposed protocol should ensure that if an attacker successfully compromises the tag's memory, he/she will not be able to trace previous communication session(s) using previously known messages.
2) Protection against replay attacks: An adversary can eavesdrop on the communications between reader and tag, obtain exchanged messages and resend these messages repeatedly. Therefore, any generated message should be fresh to the protocol session to protect against replay attacks.
3) Protection against desynchronisation: The proposed protocol should recover from desynchronisation incidents, when the attacker blocks the exchanged message(s) or when messages are lost during transmission due to system malfunction or communication error.
4) Protection against location tracking: The proposed protocol should confirm that the tag's responses are not static or linkable in order to prevent attackers from tracking the tag's location.
5) Protection against reader/tag impersonation attacks: The proposed protocol should guarantee that the reader's and tag's secret values cannot be obtained by any attacker, thus preventing an attacker from impersonating the reader or the tag.
6) Authentication: The reader and each tag in the grouping-proof should confirm their legitimacy to the server.
7) Matching: Only tags that belong to a group participate in the proof, thus reducing the time needed for the server to verify the existence of each tag.
8) Concurrency: The tags should only depend on the reader's message to respond, and should not wait for the predecessor tags messages.
9) Reading order independent: The server verifies the proof regardless of the order in which the tags were scanned.

IV. The Proposed Protocol

The proposed protocol is discussed in detail in this section.

A. System Scenario

The proposed system scenario is shown in Fig. 1, and summarised as follows:

1) The server sends an encrypted timestamp t to the reader.
2) The reader acts as a filter that separates the tags belonging to the group from the tags that do not belong to the group. It computes a *Group message* to link the tags in the group. Then, it broadcasts the *Group message* to the tags.
3) The i^{th} tag, i.e. $1 \leq i \leq n$, verifies the *Group message*, if succeeds, it sends two messages, namely the *Group message* to prove its belonging to the group, and the *Server message* to be authenticated by the server.
4) When the reader receives n responses within a pre-defined time window, it verifies the *Group message* for each tag, and generates the *Proof* containing all the received *Server messages*. Then, it sends the *Proof* to the server for validating the grouping-proof later.

B. Assumptions

We present an RFID grouping-proof protocol, which operates under the following assumptions:

- We consider a passive adversary, who has a complete control over all communications in the protocol. However,

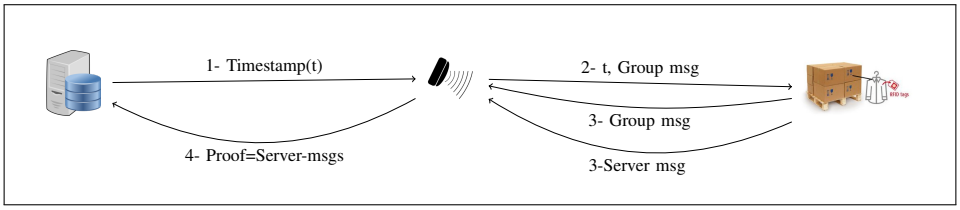


Fig. 1: System scenario

active attack, where the attacker physically tampers with the tag, is beyond the scope of this paper.

- The tag can compute XoR, generate a pseudo-random number (PRNG), and calculate hash functions. Such assumption can be implemented on an IAIK UHF RFID tag emulator discussed in detail in Section VI.
- The tags store the received readers' random numbers to avoid replay attacks.
- The reader contacts the tag through a wireless channel that is susceptible to attacks.
- The communication channel between the reader and the server is secure, for example it uses a secure HTTPS connection.
- All the operations in the tag are atomic, i.e., either all of the operations or none are processed. This mechanism protects the tags if an attacker disrupts the electromagnetic field between the reader and the tag or if the tag simply removed from the proximity of the reader.
- The groups are pre-defined and static.
- The reader is a trusted entity and tamper-resistant, for example, it has a secure memory and a rigid access control mechanism.

C. Notation

The notation used in the proposed protocol is as follows:

- The secret data stored in the i^{th} tag *(T_i)* are:
  ID_G: denotes the shared group unique identity
  TS_G: denotes the shared group secret value
  ID_i: denotes the i^{th} tag's unique identity
  TS_i: denotes the i^{th} tag's secret value
- The secret data stored in the reader (*R*) are:
  ID_i: denotes the i^{th} tag's unique identity
  K_{SR}: denotes the reader's private key
  K_{PR}: denotes the reader's public key
- Other notation includes:
  r_x: denotes a random number generated by entity *x*
  K_S: denotes the server's secret key
  H(Z): the result of generating a hash of data Z, where H: $\{0,1\}^* \rightarrow \{0,1\}^l$
  $Sign_{K_{Sx}}(Z)$: A signature on data Z, signed using entity x private key
  ts: denotes the server's timestamp
  t_n : denotes an encrypted timestamp
  n: denotes the number of tags in the group

D. Protocol Description

The proposed protocol is divided into two phases, the setup phase and the grouping-proof phase.

- Setup phase: The manufacturer's server assigns the initial values in the tags, the reader and the server. For a specific group *(G)*, each tag stores (ID_G, ID_i, TS_G, TS_i), the reader stores (ID_i, K_{SR}, K_{PR}) for all i^{th} tag belong to group *G*, and the server stores (ID_G, ID_i, TS_G, TS_i, K_S, K_{PR}) for all i^{th} tag belong to group *G*.

- Grouping-proof phase: a protocol is shown in Table I and works as follows:
  1) The server *(S)* computes encrypted timestamps t_n=E_{K_S}(ts). Each timestamp is valid for a limited time-window, where the reader should respond within this time-window. The server stores the encrypted timestamps and time-window. Since the communication channel between the server and the reader is secure, the server sends the current encrypted timestamp (t_n), ID_G and TS_G to the reader.
  2) The reader generates a fresh random number r_R, i.e. $r_R \in \{0,1\}^l$, where l is the security parameter.
  3) The reader computes two messages to link the tags chain, the first message is M_G^R=H(ID_G, r_R, TS_G), and the second message is K= $TS_G \oplus$ H($ID_G \oplus r_R$) to inform the tags about the current value of TS_G in case of a desynchronisation incident occurred.
  4) The reader broadcasts r_R, t_n, M_G^R and K.
  5) The i^{th} tag T_i generates a fresh random number r_i, i.e. $r_i \in \{0,1\}^l$.
  6) T_i re-computes M_G^R to check that it belongs to the group. If it succeeds, T_i performs the next step. If it fails, this implies either:
     a) T_i has been desynchronised in the previous session(s) resulting in not updating the value of TS_G. In this case, it needs to obtain the current value of TS_G from message (K). If succeeds, T_i performs the next step.
     b) T_i does not belong to the group if it fails to re-compute message (K), therefore it aborts the session.
  7) T_i calculates: M_i= H(ID_i, r_i, r_R, TS_i, t_n) to be included in the grouping-proof and verified by the server.
  8) T_i calculates: M_G^i= H(ID_G, r_i, r_R, TS_G, ID_i) to prove to the reader that it belongs to the group.
  9) T_i updates $TS_i^{j+1} \leftarrow$ H(TS_i^j), where *j* is the current session, and $TS_G^{j+1} \leftarrow$ H(TS_G^j) to be used in the next session *j+1*.
  10) T_i sends r_i, M_i and M_G^i to the reader. These steps are performed to each i^{th} tag in the group until the reader receives *n* responses.
  11) When the reader receives *n* responses within a pre-defined time window, it re-computes the received messages (M_G^n) for all *n* tags to confirm that only tags belonging to the group are included in the proof.
  12) For each tag belonging to the group, the reader generates *Proof* containing the received messages (M_i ... M_n), i.e. Proof= $Sign_{K_{SR}}$(t_n, r_R, r_1 ... r_n, ID_G, $M_1 \oplus ... \oplus M_n$), and then sends it to the server.
  13) Later, the server verifies the reader's signature, checks the timestamp and retrieves tags' data based on the value of ID_G. Then, it computes the expected grouping-proof (*Proof′* = $M_1 \oplus ... \oplus M_n$) regardless of the order the tags were scanned, and compares the result of *Proof′* with the received value of ($M_1 \oplus ... \oplus M_n$) in *Proof*. If there is a match, the server believes that all the tags in the grouping-proof are present and legitimate.
  14) The server updates: $TS_i^{j+1} \leftarrow$ H(TS_i^j) for all the legitimate tags in the group, and updates $TS_G^{j+1} \leftarrow$ H(TS_G^j).

TABLE I: The proposed grouping-proof protocol (successful run)

Server	Reader	T_i
1- Generates t_n		
	$\xrightarrow{t_n, ID_G, TS_G}$	
	2- Generates r_R 3- Computes: M_G^R=H(ID_G,r_R,TS_G) K= $TS_G \oplus$ H($ID_G \oplus r_R$)	
		$\xrightarrow{r_R, t_n, M_G^R, K}$
		5- Generates r_i 6- Computes M_G^R (and K⊕H($ID_G \oplus r_R$) in case of no-matching) 7- M_i= H(ID_i,r_i,r_R,TS_i,t_n) 8- M_G^i= H(ID_G,r_i,r_R,TS_G,ID_i) 9- Updates: $TS_i^{j+1} \leftarrow$ H(TS_i^j) $TS_G^{j+1} \leftarrow$ H(TS_G^j)
		$\xleftarrow{r_i, M_i, M_G^i}$
	11- Waits for *n* responses Re-computes M_G^n 12- Generates Proof= $Sign_{K_{SR}}$(t_n,r_R,r_1...r_n,ID_G,$M_1 \oplus ... \oplus M_n$)	
	$\xleftarrow{Proof}$	
13- Verifies reader and Checks: *Proof′*=$M_1 \oplus ... \oplus M_n$ 14- Updates: $TS_i^{j+1} \leftarrow$ H(TS_i^j) $TS_G^{j+1} \leftarrow$H(TS_G^j)		

V. Protocol Analysis

In this section, we analyse the proposed protocol in terms of both informal, and formal analysis using Scyther tool [18].

A. Informal Protocol Analysis

Before start analysing the proposed protocol, there are some incidents that might occur, and hence changing the execution of the protocol such as:

- Missing tag(s): If the reader does not receive a response from the 4^{th} tag for example, the reader informs the server about the missing tag by including the missing tag's ID_4 value, which uniquely identifies the 4^{th} tag, within *Proof*. The reader generates Proof= $Sign_{K_{SR}}$(t, r_R, r_{n-1}, ID_G, $M_1 \oplus ... \oplus M_{n-1}$, ID_4), then, sends *Proof* to the server. The server retrieves the missing tag data based on ID_G and ID_4 values, and calculates *Proof′* without taking into account the 4^{th} tag. The server should also alert the system about the missing tag.
- Generating an incomplete proof: If the reader receives more than the expected *n* tags (for instance, the attacker sends random messages), after a specified time-window, the reader will generate *Proof* without containing all the legitimate tags' messages. However, in this case, we recommend that the reader continues the verification process after sending the first message *Proof* and then notifies the server about the missed legitimate tags' messages that should be included in *Proof*. For example, the reader computes another message $Proof_2$= $Sign_{K_{SR}}$(r_R, r_i, ID_G, M_i, ID_i), where r_i and M_i were received after sending *Proof*, and sends $Proof_2$ to the server. Later, the server verifies $Proof_2$ first and authenticates the missing tag(s) separately, then computes the expected grouping-proof *Proof′* without taking into account those tags within $Proof_2$.

The proposed protocol attempts to meet

the goals discussed in Section III as follows:

1) Forward secrecy: The values of TS_i and TS_G are updated after each run in order to prevent forward secrecy invasion using a hash function that is irreversible. If an adversary compromises the tag T_i memory, he/she will not be able to trace the previous communications of the tag as $(M_i)^{j-1}$, $(M_G^i)^{j-1}$, $(M_G^R)^{j-1}$ and $(K)^{j-1}$ involve the use of previous secret values TS_i and/or TS_G, which are not stored in the tag. The stored updated values of TS_i and TS_G are used in the calculation of the next session and cannot be irreversible as a result of using a hash function. Hence, the attacker cannot recompute the previous messages.
2) Protection against replay attacks: All the messages contain random numbers to prevent the probability of launching a replay attack successfully. The tags should confirm first that the received reader's random number is not received previously. The inclusion of the random numbers in the messages is vital to confirm that the messages are intended for a specific reader or tag, which originally generate r_R and r_i respectively. Moreover, since r_i and r_R are fresh random numbers, it is impossible for the attacker to predict them in the next session.
   The messages M_G^R and K sent from the reader involve the reader's random number (r_R) and an updated value of (TS_G), so if the attacker re-sends the previous messages of $(M_G^R)^{j-1}$ and $(K)^{j-1}$, the tag will not authenticate these messages as the tag has received r_R within $(M_G^R)^{j-1}$ and $(K)^{j-1}$ before, and the tag's updated value of TS_G will not match the old TS_G within $(M_G^R)^{j-1}$.
   Regarding the timestamp, the message M_i contains a timestamp that is encrypted by the server to avoid predictable timestamps and is valid for a limited time during the session. Hence, the attacker cannot resend the previous M_i message as the timestamp will not be valid.
3) Protection against desynchronisation incidents: There are two scenarios, where a desynchronisation of data might occur, but the proposed protocol tackles these incidents as follows:
   - If the attacker blocks the i^{th} tag messages (r_i, M_i, M_G^i), the server will not update TS_i causing a desynchronisation of data in the next session. However, in the case of no matching, we suggest that the server computes a new value of TS_i to find a match until it reaches an upper limit, which can be set by the system owner. If the upper limit is reached, the server will not authenticate T_i, and sends a warning to the system owner about the faulty tag.
   - If the attacker blocks the reader's messages from reaching the i^{th} tag T_i, the server will update TS_G while T_i's TS_G value will remain the same, thus causing a desynchronisation in the next session. However, our protocol can prevent this attack as the reader computes an additional message (K) that contains the current updated value of TS_G received from the server, and can only be obtained by the legitimate tag belonging to the group (based on ID_G).
4) Tag's location tracking: The tags' messages involve fresh random numbers, timestamps, and/or updated TS_i and TS_G values, which means that the responses are not static. Hence, the attacker will not be able to track the tag's location. Moreover, even if the i^{th} tag does not update its values, the tag's messages will not be static

due to the existence of tag's random number (r_i) and/or timestamp.

5) Protection against tag impersonation attack: The adversary needs to be in possession of secret values such as ID_i, ID_G, TS_i, and TS_G, which are not sent in clear, in order to impersonate or clone T_i.
6) Protection against reader impersonation attack: The attacker has to have a valid digital signature private key to impersonate a legitimate reader. Moreover, the attacker cannot replay the previous messages to the tags and impersonate the reader as all the messages contain fresh random numbers, and secret values only known to the legitimate reader.
7) Authentication: The reader verifies the existence of each tag in the group based on M_G^n messages, and the server verifies the legitimacy of each tag in the group based on M_i^n messages.
8) Matching: Our protocol ensures that only the tags that belong to the group participate in the proof via the reader.
9) Concurrency: Reducing time delay by allowing all the tags to compute their messages without waiting for the other tags' responses; they send their responses when they receive the reader's messages.
10) Reading order independent: The reader generates the proof using an XoR operator, so all the tags' responses can be verified in random order regardless of the order in which the tags were scanned.

Inspired by [12], Table II demonstrates a comparison between our protocol and the other yoking/grouping-proof protocols in terms of security. As shown in Table II, the proposed protocol protects the system from different attacks, and provides forward secrecy. Sundaresan et al. protocol [5] provides similar protection but it has major issues in terms of performance as shown in Table III. In Sundaresan et al. protocol the number of rounds is proportional to the number of tags, the tags need to compute 12 PRNG functions and the server needs to perform *O(n)* computing operations to identify the tags in the group, which implies that their protocol is large in terms of tag's computing cost and server scalability respectively, making their protocol heavy to implement.

Inspired by [12] and [14], Table III illustrates the performance of the proposed protocols compared with the other grouping-proof only. We took into account the performance of the RFID tags not the reader. Our protocol is efficient for the following reasons: Firstly, it enhances performance by engaging the tags in two rounds only as in Moriyama et al. protocol [16]. However Moriyama et al. protocol does not provide forward secrecy; they assumed it is an open problem. Also, any attacker can impersonate the reader and causes the protocol to fail as the reader does not provide any proof of its identification to the tag and the server. Secondly, our protocol provides concurrency and reading order independent features. Hence, reducing time delay and failure rates respectively. Thirdly, the performance of our protocol is appropriate for RFID tags as it does not require mass memory storage; it stores four values each of which is 224-bit length, and requires average computing time cost as shown in Section VI . Fourthly, regarding the server scalability, the server retrieves the tags' data based on the value of group ID (ID_G), hence the computing cost for identifying a tag in the server is *O(1)*. In Chien et al. protocol the server computing cost is also *O(1)*, but privacy is not a concern; identifiers are sent in clear over the channel and are included in the proof. This indicates that the proposed protocol is scalable in the server side unlike the other protocols.

B. Mechanical Formal Protocol Analysis

This section presents the mechanical analysis of the proposed protocol using Scyther tool [18]. The aim of this section is to prove that the data exchanged between

TABLE II: Security comparison of yoking/grouping-proof protocols

	Juels	Saito	Piramuthu	Chien	Bolotny	Burmester	Ma	Sundaresan	Lien	Lopez2	Moriyama	Sec. IV
Forward secrecy	×	×	×	×	×	×	×	✓	×	×	×	✓
Replay attack	×	×	×	×	×	✓	✓	✓	✓	✓	✓	✓
Tag imper.	×	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓
Reader impers.	×	×	✓	×	✓	×	✓	✓	✓	✓	×	✓
Tracking	×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓

TABLE III: Performance comparison of grouping-proof protocols

	Chien	Bolotny	Burmester	Ma	Sundaresan	Lien	Moriyama	Sec. IV
# Rounds	n	n	n	n	n	n	2	2
# Messages	5	5	11	5	4	8	4	3
# Hashing	-	2	2	-	-	1	-	5*
# PRNG	3	-	-	12	12	1	3	1
# stored data	2	3	4	4	9	2	3	4
Server tags' identification	O(1)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(1)
Reading order independent	×	×	×	×	×	✓	×	✓
Concurrency	×	×	×	×	×	×	✓	✓

*The i^{th} tag might compute 6 hash functions in case it has been desynchronised

the tag and reader is secure, not to prove the authenticity of the generated proof. We omitted the message K from the script as Scyther does not check for desynchronisation incidents.

1) Mechanical Formal Analysis Using Scyther Tool: Scyther [18] is a mechanical formal analysis tool that analyses protocols under the assumption of perfect cryptography. Perfect cryptography means using cryptographic functions in the encryption process, where the adversary learns nothing from the encrypted messages unless he/she knows the decryption key. Scyther checks the secrecy and authenticity of the transmitted data. Unlike other formal tools, Scyther can verify protocols for an unbounded number of sessions, and all attacks found are actual attacks on the model [19]. Scyther has proved its capability in finding vulnerabilities in many protocols, such as in [20]–[22]

Scyther accepts the protocol description as input and outputs a summary report and displays a graph if there is an attack on the protocol. Scyther can verify protocols with an unbounded number of sessions. In Scyther, the intruder is modeled via using the channel (dy) [23]. The threat model is defined as an intruder, who has full control over the network, such that all messages sent by agents can be eavesdropped by the intruder. Moreover, the intruder may intercept, analyse, modify messages, and/or send any message he/she composes to other agents pretending to come from a legitimate agent.

The script is shown in [24]. There are three roles in the script including a server (S), reader (R), and an i^{th} tag (Ti). Each role defines its variables of type timestamp, nonce, data and ticket. *Ticket* is used to substitute any unknown term for the entity. There are two events namely, the send event (send) and the receive event (recv) followed by a sequence number; each *send* event will have a corresponding *recv* event in the other role. When specifying the variables and the event, each role then specifies the goals that the protocol attempts to meet. These goals are within the *Claim* section. In the protocol, there are four *Claim* types:

- Secret: After a successful run of the protocol, the values of the role should be secret.
- Alive: If B runs a protocol with A and it is successfully completed by a role

B, then role A has previously been running the protocol.
- Niagree (non-injective agreement): Means that A and B agreed that both role are alive and agreed to the values of the variables. Moreover, non-injective implies that several runs of B may correspond to the same run of A.
- Nisynch: Means the two events (send and recv) must be executed in the expected order as specified in the specification.

After running the script, Scyther checks the protocol based on the *Claim* requirements and did not find any feasible attack(s) within bounds, which means no attack was found within the bounded state space.

VI. Implementation and Performance Measurement

The aim of this section is to measure the performance of the proposed protocol on a resource restricted device such as an RFID tag. The main measurements are: (a) the tag's memory space, (b) the tag's computing time cost and (c) the communication cost between the reader and the tag. Communication cost refers to the time cost for a tag to read/write data in its memory and the time cost for a reader to read/write data in the tag's memory.

A. Lab Setup

Our proposed system consists of an EPC Gen2-compliant RFID tag [25], an EPC Gen2-compliant RFID reader [26], an AVR ICE JTAG programmer and one laptop (host computer) as shown in Fig. 2 (a).

For the tag side, we used a programmable, battery-powered RFID tag called *DemoTag* developed by IAIK TU Graz [25] shown in Fig. 2 (b). *DemoTag* is used to emulate real RFID EPC Gen2 passive tag and designed to add some custom commands to the EPC Gen2 standard. The tag's PCB board consists of an UHF antenna, an analogue front end and a programmable Atmel ATMega128 microcontroller.

The original *DemoTag* firmware provides an implementation of the EPC Gen2 protocol, which is implemented as a firmware library. The firmware running on the *DemoTag* is written in C code via the Crossworks for AVR IDE from Rowley Associates [**?**]. The *DemoTag* has 128 KB of flash memory, 4 KB of RAM, and 4 KB of non-volatile EEPROM memory. The firmware is stored in the flash memory, while the data such as tag's ID and messages are stored in the EEPROM. The tag stores one word (16 bits) per memory bank.

We modified the original tag's firmware by adding more functions to conform with our protocol. For example, adding functions for calculating the hash on the tag's data, and updating the tag's values. The updated firmware is debugged then uploaded to the flash memory of the microcontroller in the form of .hex via the *AVR ICE JTAG programmer*.

For the reader side, we used the *Slate (model R1260I) desktop reader* developed by CAEN [26]; it is an UHF RFID EPC Gen2-compliant reader with integrated antenna. Slate reader is embedded with an EPC Gen2 reader firmware, which is controlled by the host computer via a USB link. For the generation of the reader's random number and messages, we used Microsoft Visual Studio C#.

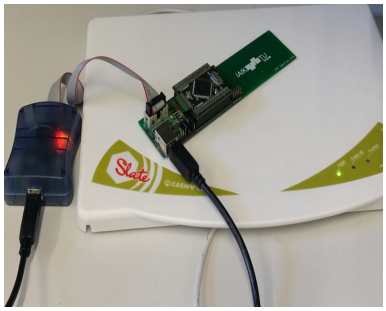
Finally, an RFID wireless link is established between the *DemoTag* and the reader.

B. Implementation process

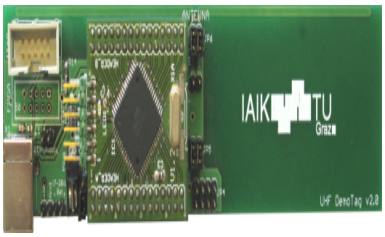
Recall that we are implementing the proposed protocol on the tag and reader sides, the server side is not implemented as it is not classified as a resource restricted device.

The tag is provided with four 224-bit secret values namely ID_G, TS_G, ID_i and TS_i, which are stored in the tag's EEPROM.

Before explaining the implementation process, it is worth to discuss the communication overhead between the reader and tag. The reader cannot write the messages



(a) The CAEN Slate Reader, the AVR JTAG ICE Programmer and the RFID UHF Demo-Tag



(b) The IAIK Graz UHF DemoTag

Fig. 2: System tools

in the tag's memory in a single write command as the tag's is programmed to write only one word (16 bits) of data in a single write command. Therefore, in our experiment, although each message is 224 bits long, we had to send the first 40 bits of each message (r_R, t_n, M_G^R and K), so the reader generates 10 write commands only; otherwise, the reader has to send 56 write commands to write the whole messages in the tag's memory, and this will slow down the communication performance dramatically.

The reader starts by generating 224-bit random number (r_R) and 224-bit timestamp (t). Then, it computes two messages, namely M_G^R and K each of which is 224-bit length (28-byte). Due to communication overhead, the reader sends the first five bytes of r_R, t, M_G^R and K to the tag in 10 WriteTagData commands to be written in the tag's memory. After the reader's messages are written in the tag's memory, the tag generates a random number R_i (224-bit), re-computes M_G^R, if there is a match with the received first five bytes, it computes two messages namely M_i and M_G^i, each of which is 224-bit length (28-byte), and places (R_i, M_i, M_G^i) in the EEPROM on the microcontroller ready to be sent by demand. Then the tag updates its EEPROM with new values of TS_i and TS_G. Later, the reader issues 3 ReadTagData commands to the tag to read the whole bytes of R_i, M_i, and M_G^i, which were stored in the EEPROM; giving 672 bits in total.

TABLE IV: Communication cost

	Tag	Reader
Read	0.07ms	327ms
Write	868.4ms	2.6 sec

To generate a random number in the tag, we used the existing PRNG function that is included in the original firmware. For the hash function, we used a SHA-2 (SHA-256) included in Crypto-avr-lib [27]. This library provides special implementations of cryptographic functions in C, which respect the extreme limited resources of microcontroller applications.

Regarding the reader, we used C# to program the reader's application. The reader's library is imported for communicating with the tag, reading the tag's memory and writing data into the tag's memory. This library does not provide methods for generating random numbers and calculating hash functions. Therefore, to generate random numbers, we use .NET Framework for this purpose, and to calculate hash on the tag's data we imported the same Crypto-avr-lib SHA-256 library for the calculation of M_G^R, K and M_G^i. For the generation of *Proof*, the reader's program uses RSA for digitally signing the grouping-proof provided by .NET Framework.

C. Performance Measurement

The performance measurements are as follows:

TABLE V: Time cost of hashing operations (milliseconds)

	M_G^R	M_i	M_G^i	TS_i	TS_G
SHA-256	13.8	13.7	13.7	13.7	13.7
Total	68.6				

1) *DemoTag* memory cost: In the proposed protocol, the memory cost is:
   - 348 bytes used from 4 KB EEPROM memory for storing tag's data, reader's messages, tag's messages, random numbers and timestamp.
   - 94 KB used from 128 KB Flash memory to store the tag's firmware.
2) Communication cost is shown in Table IV: According to [28], the time cost for the tag to read one word (16 bits) from its memory is *0.007 ms*. Hence, for the tag to compute messages, the time required to read r_R, t, M_G^R and K that were written in the tag's memory and are 160 bits in total is:

   10 * 0.007 = 0.07 ms (160 bits /16 bits = 10)

   The time cost for the tag to write one word (16 bits) in its memory is *16.7 ms*. Hence, the time required to write r_R, t, M_G^R, K, R_i, M_i and M_G^i values, which are 832 bits in total is:

   52 * 16.7 = 868.4 ms (832 bits / 16 bits = 52)

   Regarding the reader, we found that the reader needs *2.6 seconds* to write 20 bytes (160 bits) into the tag's memory in 10 write commands, and *327 ms* to read R_i, M_i, and M_G^i (672 bits) in 3 read commands, each of which reads 224 bits.
3) *DemoTag* computing cost: In a successful run of the proposed protocol, the tag computes five SHA-256 functions. Table V shows that the time cost of running five SHA-256 functions on the DemoTag is around *68.6 ms*.
   To calculate the total time for the tag's response to the reader's query, we used the equation in VI.
   Hence, the total cost for running the whole protocol on the tag is around *938 ms*, which means that the tag can respond to the reader's query in less than a second, and this relatively demonstrates the efficiency of the proposed protocol.

According to [29], the current design of the EPC-Gen2 standard on the CAEN reader has negatively affected the communication performance as the reader has to perform a fresh singulation protocol each time a tag is accessed, even if the tag is already has been accessed before. Moreover, the *DemoTag* firmware is designed to write only 16-bit word for each write command and this has slew down the communication performance in our protocol, although we send only the first 5 bytes of each value.

VII. Conclusion

The challenges in designing a grouping-proof protocol reside in the absence of the server during the scanning process. To this end, we designed a grouping-proof protocol that tackles these challenges with a low probability of delay in the responses as the tags response in two rounds and do not need to wait for the predecessor tags messages, and with a low probability of failure rate as the server verifies the proof regardless the order in which tags were scanned. Our solution also improves existing related work by protecting the system from different attacks and providing forward secrecy, which is assumed to be an open problem. To prove that the proposed protocol achieves secrecy and authenticity on the transmitted data, we formally analysed our protocol using Scyther and it did not identify any feasible attack(s). Finally, we implemented the proposed protocol to measure the performance of the tag's memory, communication and com-

TABLE VI: *Demotag* response time

T_response	= T_read +T_R2,bitwise+ T_hash + T_write
T_response	= 0.07 + 0.96 + 68.6 + 868.4 = 938 ms

puting resources, and the results showed that the protocol can be implemented with relatively low memory space and low computing time cost.

REFERENCES

[1] K. Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2003.

[2] EPCglobal, "EPC radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 MHz-960 MHz," 2008.

[3] A. Juels, "Yoking-Proofs for RFID tags," in *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, 2004, pp. 138–143.

[4] L. Bolotnyy and G. Robins, "Generalized Yoking-Proofs for a group of RFID tags," in *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual International Conference on*. IEEE, 2006, pp. 1–4.

[5] S. Sundaresan, R. Doss, S. Piramuthu, and W. Zhou, "A robust grouping proof protocol for RFID EPC C1G2 tags," *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 6, pp. 961–975, June 2014.

[6] D. Moriyama, "Provably secure two-round RFID grouping proof protocols," in *RFID Technology and Applications Conference (RFID-TA), 2014 IEEE*. IEEE, 2014, pp. 272–276.

[7] P. Peris-Lopez, J. Hernandez-Castro, and T. Li, *Security and Trends in Wireless Identification and Sensing Platform Tags: Advancements in RFID*. Hershey, PA 17033, USA: IGI Global, 2013, iSBN: 9781466619906.

[8] Y. Lien, X. Leng, K. Mayes, and J.-H. Chiu, "Reading order independent grouping proof for RFID tags," in *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on*. IEEE, 2008, pp. 128–136.

[9] J. Saito and K. Sakurai, "Grouping proof for RFID tags," in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, vol. 2. IEEE, 2005, pp. 621–624.

[10] S. Piramuthu, "On existence proofs for multiple RFID tags," in *Pervasive Services, 2006 ACS/IEEE International Conference on*. IEEE, 2006, pp. 317–320.

[11] H.-Y. Chien, C.-C. Yang, T.-C. Wu, and C.-F. Lee, "Two RFID-based solutions to enhance inpatient medication safety," *Journal of Medical Systems*, vol. 35, no. 3, pp. 369–375, 2011.

[12] P. Peris-Lopez, A. Orfila, J. C. Hernandez-Castro, and J. C. Van der Lubbe, "Flaws on RFID grouping-proofs. Guidelines for future sound protocols," *Journal of Network and Computer Applications*, vol. 34, no. 3, pp. 833–845, 2011.

[13] M. Burmester, B. De Medeiros, and R. Motta, "Provably secure grouping-proofs for RFID tags," in *Smart Card Research and Advanced Applications*. Springer, 2008, pp. 176–190.

[14] C. Ma, J. Lin, Y. Wang, and M. Shang, "Offline RFID grouping proofs with trusted timestamps," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 674–681.

[15] S. Sundaresan, R. Doss, and W. Zhou, "Zero knowledge grouping proof protocol for RFID EPC C1G2 tags," *Computers, IEEE Transactions on*, vol. 64, no. 10, pp. 2994–3008, 2015.

[16] D. Moriyama, "A provably secure offline RFID yoking-proof protocol with anonymity," in *Lightweight Cryptography for Security and Privacy*. Springer, 2015, pp. 155–167.

[17] N. Bagheri and M. Safkhani, "Secret disclosure attack on Kazahaya, a yoking-proof for low-cost RFID tags," *IACR Cryptology ePrint Archive*, vol. 2013, p. 453, 2013.

[18] C. J. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*. Springer, 2008, pp. 414–418.

[19] C. Cremers, *Scyther: Unbounded Verification of Security Protocols*. ETH, Department of Computer Science, 2007.

[20] A. M. Taha, A. T. Abdel-Hamid, and S. Tahar, "Formal verification of IEEE 802.16 security sublayer using Scyther tool," in *Network and Service Security, 2009. N2S'09. International Conference on*. IEEE, 2009, pp. 1–5.

[21] A. Mathuria, G. Sriram *et al.*, "New attacks on ISO key establishment protocols," *IACR Cryptology ePrint Archive*, vol. 2008, p. 336, 2008.

[22] A. K. Ranjan, V. Kumar, and M. Hussain, "Security analysis of TLS authentication," in *Contemporary Computing and Informatics (IC3I), 2014 International Conference on*. IEEE, 2014, pp. 1356–1360.

[23] D. Dolev and A. C. Yao, "On the security of public key protocols," *Information Theory, IEEE Transactions on*, vol. 29, no. 2, pp. 198–208, 1983.

[24] S. Abughazalah, *Scyther Script*, https://www.dropbox.com/s/z0t183z7cqq7f5h/group4.spdl?dl=0.

[25] M. Aigner, T. Plos, and S. Coluccini, "Secure semi-passive RFID tags-prototype and analysis," Bridge Project, Tech. Rep., 2008.

[26] CAEN, *R1260I - Slate*, http://www.caenrfid.it/en/CaenProd.jsp?idmod=753&parent=73.

[27] *AVR-Crypto-Lib*, https://www.das-labor.org/wiki/AVR-Crypto-Lib/en.

[28] K. Chiew, Y. Li, T. Li, R. H. Deng, and M. Aigner, "Time cost evaluation for executing RFID authentication protocols," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[29] A. Arbit, Y. Oren, and A. Wool, "Toward practical public key anti-counterfeiting for low-cost EPC tags," in *RFID (RFID), 2011 IEEE International Conference on*. IEEE, 2011, pp. 184–191.