

Persistent Security for RFID^{*}

Mike Burmester and Breno de Medeiros

Computer Science Department
Florida State University
Tallahassee, FL 32306
{burmeste,breno}@cs.fsu.edu

Abstract. Low-cost RFID tags are being deployed to support smart environment and other ubiquitous applications, and in particular to provide security and integrity functions within this domain. While the tags themselves are often discardable and easily replaced, they are embedded into a long-lasting computing infrastructure. As such, RFID security requirements may include persistence that extends beyond the lifetime of the devices.

In this paper, we discuss security mechanisms that can be used to achieve lasting security for RFID applications. The basic requirements for security include: availability (resistance against disabling attacks), authentication (unforgeability, freedom from replay attacks), and for some applications, privacy (anonymity). Additionally, for persistent security, forward-secrecy is desirable or needed. We discuss mechanisms that can be analyzed within a formal security model that allows for concurrent and composable executions. Naturally, emphasis is also placed in the practical aspects of the solutions, considering the unique characteristics of this technology.

1 Introduction

Radio Frequency Identification (RFID) tags are being massively deployed in several application and business domains to provide limited environmental awareness to computer systems. This in turn allows automation and streamlining of previously labor-intensive control processes, such as access control, authentication, shipment tracking, inventory and logistics, payment, etc. In addition to track and identifying goods, supplies, and equipment, some of these deployments are used to track and identify people, for instance RFID-enabled passports, air tickets, and implanted medical devices. Business increasingly use RFIDs to extract intelligence from operations that can contribute to their competitiveness and efficiency (e.g., the newly proposed Walmart initiative to track recyclable components). Finally, RFIDs are increasingly being considered for convenience and added-value applications for users.

As business, government, and consumer applications become more dependent on RFID-provided data for the integrity of their configuration and management, the functional integrity of the RFID tags becomes a critical requirement. While much attention by researchers has focused on the efficiency, authentication, and privacy aspects (all

^{*} This material is based on work supported by NSF award 0209092, the U.S. Army Research Laboratory and the U.S. Research office under contract number DAAD 19-02-1-0235.

fundamental concerns), it is also important to support availability—so that tags remain valid components for the duration of their projected life-time—and forward-security. The latter may not appear at first an important requirement for RFIDs, with their limited life-cycle spans. However, in the measure that RFIDs are components of larger, persistent systems, it becomes important to look at the overall picture and consider whether it is important for the system to tolerate events such as key-compromise of RFID tags.

In this paper we describe a security formalization approach that can guarantee simultaneous modeling and provision of the multiple security requirements that characterize realistic RFID usage. This approach is simulation-based and guarantees security under concurrent executions and in composition with other applications/ protocols. We also describe highly efficient protocols that are provable security within this framework.

1.1 Previous work

The research literature in RFID security, including anonymous authentication protocols, is already quite extensive and growing—for reference, a fairly comprehensive repository is available online at [?]. However, few works on RFID protocols consider security in a unified model (for examples, see [?,?]), in addition to [?]. We note that Juels and Weiss [?] propose an alternative anonymity definition following a traditional adversary-game approach (i.e., without consideration for composability issues).

In this paper, we define security in terms of indistinguishability between real and ideal protocol simulations, an approach first outlined by Beaver [?,?,?], and extended by Canetti as the universal composability framework [?,?,?]. A similar approach has also been pursued by Pfitzmann and Waidner [?,?], under the name *reactive systems*.

2 Security Context in RFID

It is well recognized that the greatest challenge in the provision of integrity and other security services for RFIDs rests on the scarcity of resources available in this computing platform. RFID protocols must consequently be lightweight and restrict themselves to the constraint envelope defined by limitations on the available power (induced by the antenna), the computational capabilities (number of cycles/second), the memory size (typically a few hundred to a few thousand bits) and the IC design (the number of gates). In particular, most RFID platforms can only implement highly optimized symmetric-key cryptographic techniques.

While recognizing the significant (and even fundamental) challenges of securing RFIDs from physical attacks, such as jamming, collision, and side-channel exploitation attacks—see, for instance the electromagnetic emanations/power consumption attacks demonstrated by Oren and Shamir [?]*—*we chose to focus on the protocol layer.

A important concern in our solution design is to accommodate features of the RFID application space. Modularity of security is important because RFIDs are components of larger applications and solutions, and therefore protocols for RFID should ideally be analyzed for security in a composable framework that allows for re-usability within different container environments. Similarly, RFID applications are designed for large scale concurrency—e.g., readers are designed to simultaneously engage with hundreds

of tags, as specified in the latest standards [?,?]—so it is important to consider concurrency issues that might affect security. We achieve this type of security by formalizing and analyzing the security of protocols within the *universal composability* (UC) framework, proposed by Canetti [?,?,?]. There are several RFID protocols that achieve this level of security by using lightweight cryptographic mechanisms [?,?]. We shall discuss these in more detail in the following sections.

2.1 RFID system entities

An RFID system involves at least three types of entities, namely *tags*, *readers* and *back-end servers*. The tags are attached to, or embedded in, objects to be identified. They consist of a *transponder* and an *RF coupling element*. The coupling element has an antenna coil to capture RF power, clock pulses and data from the RFID reader. The readers typically contain a *transceiver*, a *control unit* and a *coupling element*, to interrogate tags. They implement a radio interface to the tags and also a high level interface to a back-end server that processes captured data. The back-servers are trusted entities that maintain a database containing the information needed to identify tags, including their identification numbers, and if symmetric cryptographic primitives are employed, also cryptographic keys shared with the tags. Since the integrity of an RFID system is entirely dependent on the proper behavior of the server, it is assumed that the server is physically secure and not attackable.

In the symmetric cryptographic model, the central server can often trace and correlate all activities involving the tags. This can raise privacy concerns for the users of RFID systems, who may be wary of centralized monitoring by the operator of the central server. Correspondingly, research efforts have been dedicated to the design of privacy mechanisms that reduce the trust on the back-end server—for instance, to mitigate the ability of the server to collect user-behavior information, or to make the server function auditable. For an overview of measures and mechanisms that can be used to deal with privacy issues concerning back-end servers we refer the reader to [?]. In this paper however, we shall not investigate such privacy attacks, and instead consider the servers to be entirely trusted.

2.2 The role of forward-security

In this section, we provide several reasons why considerations of key-compromise are important in the context of RFIDs.

Forward privacy – In authentication protocols that are intended to provide privacy, forward-security implies that the privacy of earlier sessions can be preserved even after a key compromise. This is particularly important if it is possible for covert readers to exploit the vulnerability of tags to side-channel attacks, surreptitiously recovering keys—such attacks are easily deployed against current tag architectures, as shown by Oren and Shamir [?].

Key compromise detection/mitigation – To achieve forward-security in the realm of symmetric-key cryptography, it becomes necessary for keys to be updated regularly. In practice, this can be accomplished through replacement of authentication with authenticated key exchange protocols, where the older key is replaced by the newly exchanged one. Such protocols have the side-effect of revoking the older key. In the case of a secret key being compromised, there will exist two or more copies of the key, and in suitably designed schemes, these copies will diverge after interactions with legitimate readers. If the original tag is the first to then interact with a legitimate reader, the cloned keys will be revoked, automatically recovering the system to a safe state. If, on the other hand, a cloned tag interacts with the system, the original tag will be revoked, which will cause its authentication to fail in the next interaction with a legitimate reader. While an undesirable outcome, this can provide evidence of tampering. Once such tampering is detected, measures can be taken to recover the system to a safe state.

Flexibility of trust design – One of the difficulties of designing authentication protocols for RFIDs, whether or not privacy is a concern, is that symmetric cryptography—often the only type that can be implemented in tags—requires centralization of trust. A typical assumption is that the readers act as enablers, and either relay communication in real-time to a (possibly replicated) central server, or batch the interaction for later verification. However, in some cases, it would be beneficial to allow readers to store keys of tags to allow for immediate, off-line authentication. A legitimate concern in this setting is what happens after a reader is compromised, allowing the recovery of a large number of tag keys. In a protocol that does not provide for forward-security such a compromise can be very damaging. In a forward-secure symmetric-key scheme, while this situation is still a serious breach, at least the protocol provides an automated process for key revocation and update. We remark, however, that if the protocol must support large-scale key-revocation services, then forward-security mechanisms are probably insufficient—as the determination of which tag to have its key updated and which to have it revoked will require an out-of-band mechanism and private interaction—and should be complemented with a secondary secure channel, e.g., by using a second shared key dedicated to this purpose.

Mitigation of key compromise and flexibility of trust are two reasons why forward-security may strengthen security guarantees for the lifetime of tags. On the other hand, forward-privacy is in general a long-term concern. For instance, if RFIDs have been used to authenticate persons, vehicles, or to track sensitive shipments of goods, even if the keys are compromised after the end of the tag lifetime—e.g., by improper discarding of the tag—it may lead to re-construction of a person’s or vehicle’s movements in the past, or to reconstruct the distribution path of a strategic shipment. Therefore it may be important for the system to provide for security “beyond the grave,” which is the primary rationale for use of forward-secure mechanisms.

2.3 Supporting availability

Forward-security is only one of the aspects that we subsume under the label of *persistent security*. The other, and is many ways more critical, aspect of persistent security is to

ensure that the tags will not be disabled through interaction with malicious readers, and will be able to perform their functions until the end of its natural life-time.

Availability is not trivial to achieve in the context of RFID authentication. For instance, privacy concerns often require that tags use pseudo-random values or other mutable pseudonyms, avoiding repeating responses that could be used by an adversary to link different interactions with the same tag. In the symmetric-key setting, these pseudonyms must somehow be synchronized with the central server to allow for efficient recognition of tags. If the adversary is somehow successful in de-synchronizing tags from the central server, the tags may be permanently disabled. Some very ingenious protocols [?,?] that provide for robust privacy guarantees are thus vulnerable to disabling attacks.

Another source of threats to availability is the common support of kill-keys. If the disabling functionality is not protected by authentication, it opens an obvious channel for an adversary to cause massive disruption through large-scale disabling of tags. It is also not always appropriate to disable a tag by simply disclosing a (long-term) shared authentication key. For instance, if the protocol does not provide for forward-security, adversarial eavesdropping of the killing interaction will compromise the security of all past tag authentication sessions.

3 The UC Framework in the RFID Application Setting

When working with RFID systems, it is important to keep in mind that they are often components of a larger ubiquitous computing architecture, and that therefore it is important to design their protocols for modular deployment, facilitating re-use in other contexts. Therefore, it is appropriate to consider security issues related to concurrent executions and to modular composition of protocols. One approach to provide for composable security properties is to formalize security within the Universal Composability (UC) framework, which we describe in this section. We also outline our contribution to the formalization and design of persistent security mechanisms for RFID, originally introduced in [?]. This includes a UC authentication framework that extends a model in [?] to include forward-anonymity and a protocol—described in Section 4—that provides for forward-anonymous authentication and that guarantees availability. A significant characteristic of this persistently secure protocol is its liveness: As the only required security primitive is a pseudo-random function, it can be implemented through various constructions that can be highly optimized, as we describe in Section 5.

3.1 UC security generalities

UC security is based on notions of interactive indistinguishability of real from ideal protocol executions. This approach requires the following components:

1. A mathematical model of real protocol executions, where honest parties are represented by probabilistic polynomial-time Turing machines (PPT) that correctly execute the protocol as specified, and adversarial parties that can deviate from the protocol in an arbitrary fashion. The adversarial parties are controlled by a single

- PPT adversary that (1) has full knowledge of the state of adversarial parties, (2) can arbitrarily schedule the communication channels and activation periods of all parties, both honest and adversarial, and (3) interacts with the environment in arbitrary ways, in particular can eavesdrop on all communications.
2. An idealized model of protocol executions, where the security properties do not depend on the correct use of cryptography, but instead on the behavior of an *ideal functionality*, a trusted party that all parties may invoke to guarantee correct execution of particular protocol steps. The ideal-world adversary is controlled by the ideal functionality, to reproduce as faithfully as possible the behavior of the real adversary.
 3. A proof that no environment can distinguish (with better than negligible accuracy) real- from ideal-world protocol runs by observing the system behavior, including exchanged messages and outputs computed by the parties (honest and adversarial). The proof works by translating real-world protocol runs into the ideal world.

In the UC framework, the protocol execution context is captured by a *session identifier sid*. The *sid* is controlled by the environment \mathcal{Z} , and reflects external aspects of execution, as for example, temporal and/or locational issues, shared attributes and/or keys, etc. All parties involved in a protocol execution instance share the same *sid*. In particular, the security proof cannot make any assumptions about extraneous knowledge that may or not be available to \mathcal{Z} through interactions with other entities (including other instances of the protocol). The environment \mathcal{Z} is the first party to become active in any simulation, and it activates the adversary next. If the adversary (and all other parties) become inactive, control passes to the environment. The adversary and \mathcal{Z} may interact in arbitrary ways, and the real-world simulation halts when the environment halts.

Observe that the ideal functionality security is unconditional, and does not rely on any cryptographically primitives that are computationally secure. This is because, in the UC framework, the security supports concurrent executions. It does not follow from this, however, that the security of the real protocol is unconditional, because we only provide for computational indistinguishability of protocol runs in the real world from their simulated counterparts in the ideal world.

3.2 The composable security model for persistent security in RFID

Entity authentication is a process in which one party is assured of the identity of another party by acquiring corroborative evidence. Anonymous authentication is a special type of entity authentication where the identities of the communication parties remain private to third parties that may eavesdrop on their communication or even invoke and interact with the parties. In the UC framework, it is captured by the parties having ideal access to an anonymous entity authentication functionality, which we denote by $\mathcal{F}_{\text{auth}}$. This functionality is presented in Figure 1.

There are two types of protocol parties, `server` and `tag`. In each session, there is a single instance of a party of type `server` and arbitrarily many instances of type `tag`. The function $\text{type}(p)$ returns the type of party p in the current session. The UC entities, such as adversary \mathcal{A} and the environment \mathcal{Z} , are not parties per se, though the \mathcal{A} may control several protocol parties. A single session spans the complete life-time

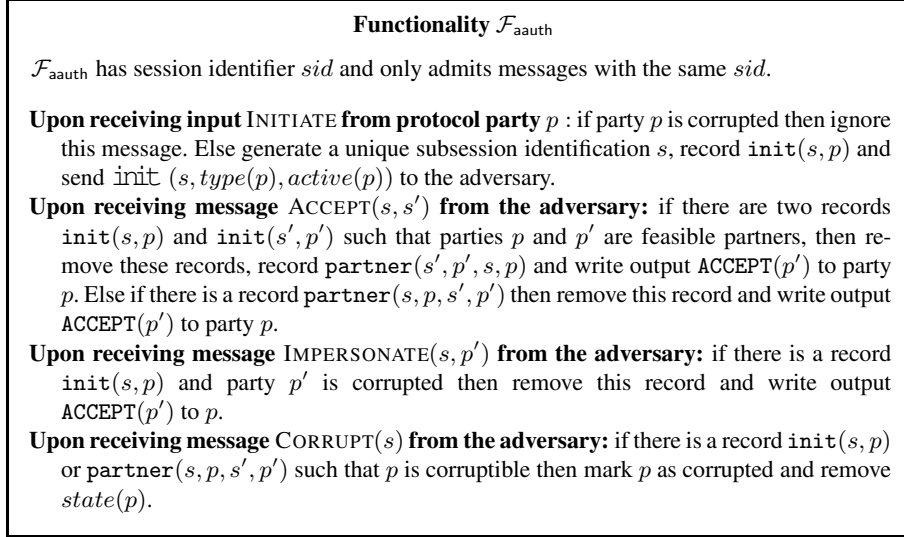


Fig. 1. Ideal anonymous authentication

(simulation instance) of our authentication scheme. It consists of many concurrent sub-sessions, which are initiated by protocol parties upon receiving input INITIATE from the environment \mathcal{Z} . While the server and tags initiate sub-sessions, the adversary controls the concurrency and interaction between these sub-sessions.

Two protocol parties are *feasible partners* in authentication if they are, respectively, a server and a tag. Upon successful completion of a sub-session, each party accepts its corresponding partner as authenticated. The environment \mathcal{Z} may read the output tapes of the tags and server at any moment during the session, which terminates when the environment \mathcal{Z} stops. The environment \mathcal{Z} may contain many other sessions of arbitrary protocols, thus allowing our protocol to start and run concurrently with arbitrary others. All parties involved in a sub-session of the authentication scheme are given a unique session identifier sid by the environment \mathcal{Z} .

A careful reading of the definition of $\mathcal{F}_{\text{auth}}$ will satisfy the reader that it simultaneously provide for the following security properties:

Authenticity: Results from invocations of the command ACCEPT, one for each partner. The true identity of the partner is given to the authenticating parties, regardless of the action of the adversary. This limits the adversary to invocation of the protocols and scheduling of the output of each party only.

Anonymity: The only information revealed to the adversary by the functionality is the type of the party, whether it is a tag or server. The difference between tag and server is observable since the real server always starts the protocol, and accordingly the ideal runs emulate this behavior.

Forward-security: The effect of corruption in the ideal world, via command CORRUPT, is that the adversary can impersonate tags, using the IMPERSONATE command. The adversary may also link all incomplete subsessions of the same corrupted party, up to the last one that completed successfully, by acquiring knowledge of $active(p)$ —the list of identifications of all preceding incomplete subsessions, obtainable via command INITIATE. However, once a subsession is successfully completed in the ideal world, this subsession and all earlier subsessions of the same party are protected against all future corruptions of any party. Therefore, the ideal world provides forward-security only for completed subsessions. Note that, in the functionality, $state(p)$ is the list of all subsession records maintained by the functionality concerning party p in the current session. This list is removed from the memory of ideal functionality upon corruption of the tag p , and effectively leaves control of the corrupted tag to the adversary. The only information retained is the fact that p is corrupted.

Availability: No interface is provided to the ideal adversary to disable tags, except by corrupting them. In the Byzantine model, the adversary can effectively always deny service by indefinitely delaying delivery of messages, and this is intrinsically available in the ideal world; however such denial-of-service requires constant and omnipresent intervention of the adversary, and is not a relevant threat in practice.

While we do not describe here the functionality for authenticated key-exchange we note that, in the context of persistent security it must provide the additional guarantee of (forward-secure) session-key indistinguishability. We refer the reader to [?] for details, and also for the security proof—showing that the protocol described in the next section realizes the functionality $\mathcal{F}_{\text{auth}}$.

4 Protocols

We now revisit O-FRAP, an {O}ptimistic, {F}orward-secure {R}FID {A}uthentication {P}rotocol introduced in [?]. The protocol is highly optimized for the case when the system is not under attack. It relies on a trusted setup, namely a secure, out-of-band initialization of the server’s database and the tags’ key-storage with shared keys for symmetric cryptographic primitives, to be described in further detail in Section 5.

These protocols are lightweight enough for RFID deployments, yet provide strong UC security and therefore are suitable in other ubiquitous application contexts, such as sensor networks. The only restriction is that the each component playing the role of a single tag must use separate keys when performing parallel authentications/key-exchanges, as we have not modeled concurrency of (honest) tag behavior.

4.1 Trusted Setup and the Server Database

The following trusted setup is done in a physically secure environment. For each tag, a fresh, unique key pair (r, k^a) is randomly generated and stored both at the tag and the server. The value r is a one-time-use pseudonym for the tag that is used for optimistic key-retrieval, while the value k^a is the tag’s authentication key (updated after each successful authentication).

The tag stores the key pair in its non-volatile (re-writable) memory, while the server initializes a database D whose entries are of the form $\langle i, previous_i, current_i \rangle$. At setup, $previous_i = (\perp, \perp)$, while $current_i = (r_i, k_i^a)$. The server must maintain two key pairs for each tag to preserve consistency though key updates in the presence of active adversaries: Since the server computes the updated triple before the tag, an adversary could tamper with the communication channel and prevent the tag from computing the updated key. During an authentication attempt by the tag i , the server detects whether the tag is using $previous_i$ or $current_i$. If the tag uses $current_i$, the server will replace $previous_i$ with $current_i$ and $current_i$ with a newly computed value. If the tag uses $previous_i$ instead, then $current_i$ is replaced with newly computed value, while $previous_i$ is preserved. This operation is denoted $D.update(i)$.

We assume that the database is (doubly) indexed by the values of the previous r_i , denoted $previous_i(r)$, and the current r_i , denoted $current_i(r)$. Therefore, database entries $\langle i, previous_i, current_i \rangle$ can be efficiently retrieved from either value. We denote this operation by $D.retrieve(r)$.

4.2 RFID entity authentication

In O-FRAP, In this protocol, r_{sys} and r_{tag} are values generated pseudo-randomly by the server and the tag, respectively, so as to anonymize the session and to prevent replays. The value r_{tag} is generated pseudo-randomly for optimistic identification of the tag, while k_{tag}^a is the tag's current key and is updated by the server after the tag is authenticated, and by the tag after the server is authenticated.

On activation by the server, the tag computes four values $\nu_1, \nu_2, \nu_3, \nu_4$ by applying the pseudo-random function F to $(k_{tag}^a, r_{tag} || r'_{sys})$. We use the following convention: If the sender writes the value x to a channel, it is observed as x' by the receiver. The value x' may differ from x if corrupted by the adversary while in transit.

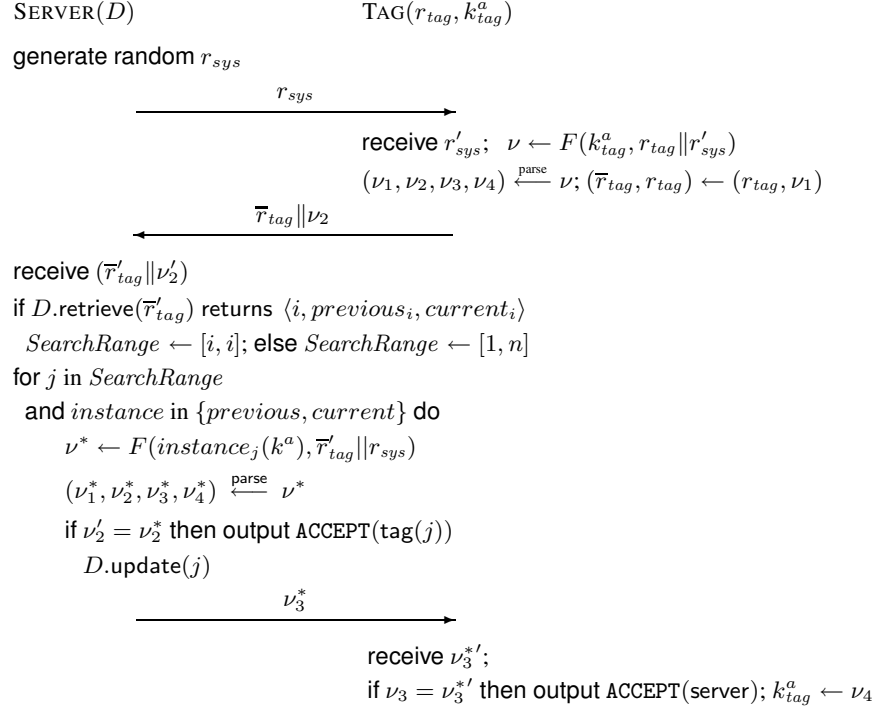
In O-FRAP, ν_1 is used to update the pseudo-random value r_{tag} ; ν_2 is used for authentication of the tag; ν_3 is used to authenticate the server; ν_4 is used to update k_{tag}^a . In our protocols we use the following convention: the four values computed by the server by applying the pseudo-random function F to $(k_j^a, r'_{tag} || r_{sys})$ are denoted by $\nu_1^*, \nu_2^*, \nu_3^*, \nu_4^*$. When the adversary is passive, these values correspond to the non-starred values. In particular $\nu_2^* = \nu_2'$ and $\nu_3^{*'} = \nu_3$, and the server and tag output ACCEPT.

Observe that the tag key k_{tag}^a is updated after each server authentication, giving strong separation properties between sessions. In particular, if a tag is compromised, it cannot be linked to transcripts of earlier sessions. This guarantees forward-anonymity.

5 Implementation details

O-TRAP requires only the use of pseudo-random functions (PRFs). This results in a very flexible architecture since a variety of well-known and validated PRF constructions are established. Efficiency vs. security trade-offs in this architecture are easily achieved, as key-size and pseudo-randomness (estimated as the logarithmic length of the PRF cycle) can be chosen to the granularity of individual bits. Here we discuss two implementation strategies based on different PRF instantiations.

Fig. 2. O-FRAP: Optimistic Forward-secure RFID entity Authentication.



Using a well-known technique by Goldreich et. al. [?], it is possible to build a PRF that makes a call to a pseudo-random generator (PRG) per bit of input processed. In turn, a very efficient PRG implementation can be achieved using linear feedback shift registers, such as the self-shrinking generator [?]. This results in a small number of bit operations per input and output bits. Moreover, the entire footprint of the implementation can be fixed to require fewer than 2K gates to achieve 128-bit security [?], a range feasible for many RFID architectures (and within the EPC class 2 constraints). A recently proposed implementation has achieved 128-bit security with only 1435 logic gates within 517 clock cycles and 64B memory [?].

Block ciphers can similarly be used to implement PRFs through a number of standard constructions [?]. When used only as PRFs, these constructions are in practice more efficient (in particular with regards to footprint) than security algorithms that require protocol parties to perform both encryption and decryption operations. Recently, a highly optimized implementation of the Advanced Encryption Standard (AES) [?] block cipher algorithm has been achieved which is suitable for RFID tags [?]. An RFID architecture using this implementation was proposed in [?], with footprint equal to 3,400 gates (in this implementation, gate complexity is based on 2-input NAND gates, called gate equivalents), and mean current consumption equal to $8\mu A$ at 100kHz and

within 1032 clock cycles. Such implementations are more efficient than achievable by hash-based protocols, as demonstrated in [?].

6 Conclusion

Despite the constraints imposed by RFID technology, it remains possible to design provably secure mutual authentication protocols that are feasible to implement and that support persistent security. The solutions described herein have been analyzed for their security within a composable, modular framework that provides confidence in their re-usability in new contexts.