# Unidirectional Key Distribution Across Time and Space with Applications to RFID Security

Ari Juels
*RSA Laboratories*
*Bedford, MA, USA*
*ajuels@rsa.com*

Ravikanth Pappu
*ThingMagic Inc*
*Cambridge, MA, USA*
*ravi.pappu@thingmagic.com*

Bryan Parno
*Carnegie Mellon University*
*Pittsburgh, PA, USA*
*parno@cmu.edu*

## Abstract

We explore the problem of secret-key distribution in *unidirectional* channels, those in which a sender transmits information blindly to a receiver. We consider two approaches: (1) Key sharing across *space*, i.e., via simultaneously emitted values that may follow different data paths and (2) Key sharing across *time*, i.e., in temporally staggered emissions. Our constructions are of general interest, treating, for instance, the basic problem of constructing highly compact secret shares. Our main motivating problem, however, is practical key management in RFID (Radio-Frequency IDentification) systems. We describe the application of our techniques to RFID-enabled supply chains and a prototype privacy-enhancing system.

## 1 Introduction

Key management is a cornerstone of cryptography, but also its major deployment challenge. Textbook cryptographic protocols often presuppose keys held by a pair of principals anecdotally dubbed Alice and Bob. From birth, Alice and Bob are presumed to share a password, a secret key, or the public key of some mutually trusted entity.

In practice, the conceptually simple goals of key distribution—even between two parties—are fraught with complexity. Disparate naming conventions and requirements for key revocation and recovery have hobbled many public-key infrastructures. Password management remains a widespread challenge thanks to obstacles as varied as limited human memory, caps-lock keys, and social-engineering attacks such as phishing.

Ultimately, key distribution must rely on secure channels established through pre-existing trust relationships or special physical considerations. For example, browser software shipped with new computing systems carries the root public keys of a number of certificate authorities. Special physical assumptions and adversarial constraints can shape the problem of key distribution in interesting ways. Researchers have explored various physical models to support key establishment between pairs of devices, including optical channels [16,24], distance-bounding [30] based on signal velocity, and physical contact [33]. Such models treat a variety of adversarial capabilities. For instance, privacy amplification [3], which strengthens keys using shared sources of noise or quantum phenomena, appeals to bounds on adversarial data access or storage.

In this paper, we focus on the problem of key distribution between two parties communicating via a *unidirectional* channel. This special constraint means that one party (Alice) acts exclusively as a sender, while the other (Bob) acts exclusively as a receiver. We consider the challenge of unidirectional key transport when Alice and Bob have no pre-existing relationship, but share a channel with limited adversarial access. We believe that such special unidirectional models have broad applicability, as they reflect the natural broadcast characteristics of many media. The starting point and motivation for our investigation, though, is the specific, real-world problem of key transport in RFID-enabled supply chains.

**Organization** In Section 2, we give details on the RFID challenges motivating our work. We provide an overview of our technical contributions in Section 3 and review related work in Section 4. In Section 5, we present what we call *secret sharing in space*, a key-distribution system that supports privacy protection in RFID applications. We also briefly describe a prototype RFID implementation of secret sharing in space. In Section 6, we present *secret sharing in time*, a separate body of techniques applicable to RFID access-control and authentication, and also of broad interest for key distribution in unidirectional channels. We conclude in Section 7 with a brief discussion of future research directions.

## 2 Motivation: The RFID Landscape

The ratio of terrestrial radio and cellular telephone systems to the number of humans on earth is approaching unity, and in the past decade, a completely different kind of radio device has emerged and is poised to eclipse this ratio by three orders of magnitude. Rapid advances in CMOS technology have enabled the production of low-cost *tags* that are capable of reporting their identity over a wireless link. These tags—usually costing tens of cents and carrying a few thousand gates of silicon—have little if any general-purpose computing power beyond what is needed to respond to commands from an interrogator or *reader*. This asymmetry between interrogators and tags is further amplified by the fact that, in many applications, tags are passive, lacking an on-board source of power; instead, they harvest power from the electric, magnetic or electromagnetic field generated by the interrogators.

Recent developments in passive Radio Frequency IDentification (RFID) technology and corresponding international standards [12] have spurred deployment in applications ranging from supply-chain and inventory management of consumer goods, to tracking medical equipment in hospitals, to counting poker chips on gaming tables.

The heir apparent to the optical barcode, RFID is becoming a prevalent technology in supply-chain management. Ultimately, manufacturers and retailers envisage RFID tagging of individual consumer *items*. Today, tagging is most common at the granularity of *cases*, which contain consumer items, and of *pallets*, which carry cases. In this paper, we use the term "case" as the generic term for a discrete collection of goods.

For supply-chain operations, the predominant RFID standard is one known as the Electronic Product Code (EPC) (in particular, Class-1 Gen-2 EPC, hereafter referred to as Gen2). EPC tags act effectively as wireless barcodes, emitting short strings of information known as EPC codes. An EPC code has four basic components: (1) A *header*, which denotes the EPC version number; (2) A *domain manager*, which typically specifies the manufacturer or creator of the item; (3) An *object class*, which specifies the item type, and (4) a *serial number*, a unique identifier for the item. This *license plate* approach associates an arbitrary amount of metadata with the tagged object while requiring little memory on the tag itself.

### 2.1 Security and Key Distribution in Gen2

Two features in the Gen2 standard require secret keys:
**Locking and perma-locking:** It is possible to lock part (or all) of the tag's memory, either temporarily under a 32-bit password, or permanently with no possibility of unlocking and rewriting the memory. While this feature prevents unauthorized entities from tampering with the contents of tag memory, it does not prevent unauthorized readers from reading the contents.

**The kill command:** The only security function that completely disables tags is a command known as *kill*. When transmitted by a reader along with a tag-specific kill PIN (32 bits long in Gen2), the kill command causes a tag to disable itself permanently.

The EPC kill function is envisaged as a privacy-enhancing feature for retail environments with item-level tagging. EPC tags specify the items to which they are affixed. Thus a consumer carrying EPC-tagged items would in principle be subject to clandestine inventorying attacks that disclose sensitive data about medications, reading materials, luxury goods, and so forth. By deploying the kill function at the point of sale, a retail shop can protect against such privacy infringements by disabling tags. Additionally, researchers have proposed anti-cloning techniques that co-opt the kill and write-access commands in EPC to support reader authentication of tags and to protect PINs from untrusted readers [15].

Both locking and killing pose a significant implementation hurdle: They require a solution to the *key-distribution* problem. The initialization of tag-specific kill PINs in tags and the secure propagation of these PINs to point-of-sale devices are formidable operational challenges. Supply chains include entities with widely disparate data-processing capabilities. Information transfer across organizational boundaries, moreover, introduces a host of regulatory and technical burdens. Hence supply-chain entities commonly lack data-network mechanisms for timely, reliable, and secure transport of PINs. While it might seem a straightforward matter for Alice (a manufacturer) to share EPC PINs with Bob (a retailer) through a data network, in practice it is often quite difficult. Indeed, with all of the intermediaries through which manufactured goods regularly pass, Alice may even ship cases without knowing that Bob is the ultimate receiver.

In this paper, we show that RFID-enabled supply chains possess unique properties that allow us to:

- Provide consumer privacy with respect to unauthorized scanning of tagged objects;
- Provide a robust protocol-independent mechanism to distribute PINs and passwords *without requiring a network connection, changes to the air interface protocol, or changes to the tag hardware*.

The only resource our method requires is memory on the tag, and we provide a means to trade-off memory usage against security.

### 2.2 Object Hierarchies in RFID-Enabled Supply Chains

Our techniques for key distribution in RFID applications rely in part on the fact that supply chains are hierarchical in nature. To highlight the properties we utilize, we use Figure 1 to trace the path of a single pack of razor-blades in a consumer's home back to the manufacturing facility.

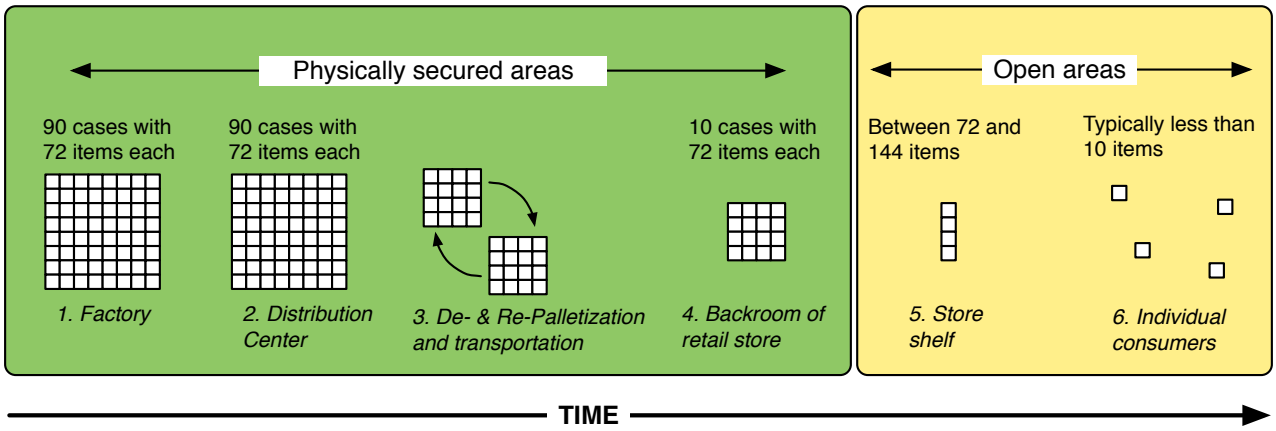



Figure 1: **Object hierarchies in RFID-enabled supply chains** *This schematic represents the path taken by an individual pack of razor blades from the factory to the consumer's home. Please refer to Section 2.2 for details.*

Typically, items start off in large collections and progressively get whittled down into smaller aggregates as they make their way from the factory to the store shelf [13]. In the example above, razor blades are assembled into a pallet containing 90 cases, each with 72 packs of blades. Assuming the items, cases, and pallet are tagged, we have a total of 6571 tags on this particular pallet. The pallet is then transported, possibly with many other pallets, to a distribution center (DC). The DC de-palletizes the large pallet and assembles a mixed pallet with a smaller quantity of cases that has been ordered by the store. A typical number of cases from the original pallet that make it onto this new pallet is 10 [13]. Assuming a new pallet tag is added, 730 of the 6571 original tags are now available on the new pallet. This new pallet is then transported to the store and stored in the backroom. Of these 730 tags, typically up to two cases' worth, or 144, items are laid out on the store shelf for customers. From this collection, consumers pick up a few packs and purchase them. Therefore, the object hierarchy is as follows.

*Razor blades*: $6571 \rightarrow 730 \rightarrow 144 \rightarrow 5$

Similarly, for DVDs a typical object hierarchy is

*DVDs*: $5040 \rightarrow 2520 \rightarrow 400 \rightarrow 24$

where the last number represents an estimate of the number of DVDs from a case sold to an individual consumer. Finally, for pharmaceuticals, we have

*Pharmaceuticals*: $7200 \rightarrow 1920 \rightarrow 150 \rightarrow 6$

where again the last number represents an estimate of the maximum number of filled prescriptions from one case in possession of a consumer at the same time.

While these numbers may vary between different types of retailers and use cases, the important point to note is that the number of tagged items starts off large and ends up being small. Another important insight is that larger numbers of tags are typically found in physically secure areas, while smaller numbers of tags are found in physical locations that are accessible to adversaries. We exploit the fact that tags share the same space-time context earlier in the supply chain, but this history is progressively lost as tagged objects emerge from the supply chain into the front of the retail store and thereon into the consumer's home.

## 3 Our Contribution

The challenges of EPC PIN distribution motivate us to consider a new approach, that of *transporting secret keys in RFID tags themselves*. This approach allows a unidirectional model of key transport. The sender (Alice) encodes secrets across tags or cases. The receiver (Bob) recovers these secrets without communicating with Alice—and, potentially, without even knowing her identity.

To support this unidirectional model of key transport, we propose protocols for dispersing keys or PINs across tags by means of *secret sharing*. We consider two distinct modes of secret sharing: (1) *Secret sharing across space* and (2) *Secret sharing across time*.

**Secret sharing across space:** Alice can share a secret key $\kappa$ across a set of tags $T = \{\tau_1, \ldots, \tau_n\}$ in a case. To do so, she transforms $\kappa$ into a collection of shares $S_1, \ldots, S_n$, and stores $S_i$ on tag $\tau_i$, such that $\kappa$ can only be recovered by scanning all $n$ tags in the cases. (We later consider threshold secret sharing, i.e., schemes such that $k < n$ shares suffice for recovery of $\kappa$.)

Such secret sharing across tags permits a new approach to privacy enforcement for item-level tagging that largely *eliminates the need for killing tags*. Suppose that $m_i$ consists of the data, e.g., EPC code, associated with tag $\tau_i$. Suppose that Alice replaces $m_i$ with $E_\kappa[m_i]$ in all tags, where $E_\kappa$ represents symmetric-key encryption under $\kappa$. Then the contents $m_i$ of any tag can only be deciphered by scanning the full set of tags $T$.

On receiving a case from Alice, a retailer (Bob) can recover $\kappa$ and decrypt the EPC codes in its tags. *Once the items and their associated tags are dispersed by sale to customers, however, a would-be eavesdropper has no practical way to recover* $\kappa$. We assume here that access to tags is secured in the supply chain, i.e., the pre-sale environment. We illustrate the principle by example.

**Example 1** *Alice ships a case containing three bottles of medicine bearing RFID tags* $\tau_1, \tau_2$ *and* $\tau_3$ *with data strings* $m_1, m_2$, *and* $m_3$. *She generates a secret key* $\kappa$ *and transforms it into a triplet of shares* $(S_1, S_2, S_3)$ *via a (3,3)-secret sharing scheme. Alice writes the value* $v_i = (E_\kappa[m_i], S_i)$ *to tag* $\tau_i$.

*Bob, a pharmacist, receives Alice's case. He scans the three tags, recovers* $\kappa$ *and decrypts the data strings of the tags in the cases, enabling him to read* $m_1$ = *"High street-value drug, 500 mg, 100 count, bottle #8278732," as well as* $m_2$ *and* $m_3$. *Bob dispenses the first bottle to Carol.*

*Later in the day, a drug thief surreptitiously scans Carol's RFID tags as she passes on the street. The thief obtains the value* $v_1 = (E_\kappa[m_1], S_1)$*—a ciphertext and key share that by themselves carry no meaning and therefore do not reveal the presence of high-value pharmaceuticals.*

As this example illustrates, Bob does not have to perform any explicit action to protect his customers' privacy. He does not have to kill or rewrite tags. Secret sharing across space enforces privacy implicitly through the physical dispersion of tags. Unlike killing, though, secret sharing does not enforce privacy against tracking attacks. The value $v_1$ is itself a unique identifier that can serve to correlate different instances of scanning of Carol's tags and potentially track Carol herself. This is a basic limitation of our scheme, but one we consider to be of considerably smaller importance than revelation of tag data contents.

Of course, it is possible to encode $\kappa$ in a case-specific tag, rather than across items within a case. The advantage of sharing across space is twofold, though: (1) As we show, it allows for robust secret recovery, i.e., recovery of $\kappa$ even in the face of scanning errors or lost data and (2) It eliminates the need for an extra tag, i.e., one on each case.

Our main research challenge in applying secret sharing across space to RFID is the development of schemes with *tiny* secret shares. While the literature on computational secret sharing considers shares of length equal to that of a secret key, e.g., 128 bits, space constraints on EPC tags urge even smaller share sizes, e.g., 16 bits.

In Example 1, the adversary (thief) is *underinformed*, i.e., lacks the shares needed to recover $\kappa$. Another facet of our research aims to create situations in which an adversary is *overinformed*, having too many shares to identify and extract tag keys. In Appendix A, we consider situations in which an adversary is overinformed when scanning retail shelves where the contents and thus RFID tags of many cases are mixed together.

**Secret sharing across time:** Suppose that $\kappa$ is not an encryption key, but a write-access key. In that case, the ability to recover $\kappa$ by scanning a case would enable a malefactor with access to a single case at any point in the supply chain to modify the data contents of tags. Similarly, suppose that $\kappa$ were a symmetric key used to authenticate tags. Then simply by scanning a case, an adversary could recover all of the key material required to clone the associated tags.

For this reason, we consider another form of secret sharing in which a secret key $\kappa$ is distributed not across the tags in a single case, but across multiple cases. Given that cases—much like data packets—depart and arrive at staggered times in a supply chain, we refer to this approach as secret sharing across time.

**Example 2** *Alice, a manufacturer, is shipping cases of RFID-tagged items to Bob. She would like to communicate the write-access PINs for the tags in these cases to Bob as securely as possible.*

*Suppose that Alice employs trucks that hold up to ten cases. She might do as follows. She selects a window, i.e., sequence, of eleven cases* $c_j, c_{j+1}, \ldots, c_{j+10}$ *designated for delivery to Bob. She creates a master secret* $\kappa$ *from which it is possible to derive the write-access PIN for any tag within the window of cases. She distributes* $\kappa$ *into eleven shares* $S_1, S_2, \ldots, S_{11}$ *via an (11,11)-secret sharing scheme, and writes share* $S_d$ *to case* $c_{j+d-1}$. *(She might distribute the secret across tags on individual items, or on a case-specific tag.)*

*An adversary that gains access to the contents of a small collection of cases, or even an entire truckload, is unable to reconstruct the secret* $\kappa$ *or to obtain the write-access PINs for the RFID tags. On the other hand, Bob can reconstruct* $\kappa$ *once he receives the full sequence of eleven constituent cases.*

Of course, in practice it may be difficult for Alice to identify *a priori* a window of cases that a legitimate receiver, Bob, will receive in its entirety, particularly if the cases pass through intermediaries. Hence the main thrust of our work here is the development of more flexible secret sharing schemes. We propose what we call *Sliding-Window Information Secret-Sharing* (SWISS) schemes, constructions such that for a sequence $c_1, c_2, \ldots$ of cases, Bob need only receive a minimal number $k$ of cases in any contiguous window of size $n$ in order to reconstruct the associated secret keys. SWISS schemes provide key confidentiality against adversaries that intercept cases on a sporadic basis.

As we explain, it is a straightforward matter to create a SWISS scheme in which shares are linear in $n$, and thus potentially large in practice. Our contribution is a SWISS scheme whose shares are constant in size, i.e., have length independent of $k$ and $n$.

## 4 Related Work

Since its invention in 1979 by Shamir [32] and independently by Blakley [4], secret sharing has played a foundational role in cryptography. However, our work differs from previous work in two key aspects: the privacy goal we adopt and the size of the shares employed.

The majority of secret sharing literature evaluates the privacy of a secret-sharing scheme from an information-theoretic perspective, seeking to create efficient schemes for various access structures. In this regime, a perfect secret-sharing (PSS) scheme is one in which an adversary learns no information about the secret in an information-theoretic sense (i.e., even if the adversary has unbounded computational resources). Shamir's scheme [32] qualifies as a PSS scheme. Statistical secret-sharing (SSS) schemes, such as Blakley's [4], allow a small amount of information leakage, in the information-theoretic sense.

A narrower literature concerns complexity (or computational) theoretic secret-sharing (CSS), in which privacy depends on computational bounds on an adversary. Krawczyk first introduced the notion of a CSS scheme [20], and Bellare and Rogaway later refined and formalized it [2]. Work in this area has focused on privacy based on *all-or-nothing* indistinguishability. In other words, in Krawczyk's construction, an adversary either has no information about the secret or she has complete information about it. In this work, we introduce constructions that accommodate *gradated* key information. This allows us to consider schemes in which the leakage of secret information is proportional to and thus grows *gradually* with the number of revealed shares.

The other dimension in which this work differs from previous work is the length of the shares involved. It is well known that in any natural PSS scheme, the size of every participant's share must be at least that of the secret itself [10, 18]. For specific access structures, stronger lower-bounds have been shown [9].

Any scheme in which shares are shorter than the secret is necessarily imperfect. Ogata and Kurosawa [26] give information-theoretic lower bounds on share sizes in such schemes. At a high level, they show that a share must have length equal to at least that of the "gap" in knowledge between sets of shares outside the permitted access structures and the secret itself. More formally, suppose that a secret $x \xleftarrow{R} D$ is selected at random from distribution $D$. Let $\hat{x}$ denote a random variable for $x$ and $\hat{S}_i$ one for $S_i$, i.e., the $i^{th}$ share generated by a natural secret-sharing scheme. If $\Gamma$ represents the set of access structures that are allowed to recover the secret, then it is the case that $H(\hat{S}_i) \geq min_{\gamma \notin \Gamma} H(\hat{x} \,|\, \{\hat{S}_i\}_{i \in \gamma})$, where $H(A \,|\, B)$ denotes the entropy of $A$ conditional on $B$.

In terms of concrete proposals, in the information-theoretic literature, McEliece and Sarwate note that Shamir's scheme can be generalized as a Reed-Solomon code, permitting a tradeoff between share size and security [25]. Blakley and Meadows propose a class of ramp secret sharing schemes [5] which define two thresholds. Given $t$ shares, it is easy to reconstruct the secret. Less than $t'$ shares reveals no information about the secret, and given some number of shares $y$ such that $t' \leq y < t$, the information gained about the secret is proportional to $\frac{y-t'}{t-t'}$. Larger "ramps" provide weaker security but allow a reduction in share size. In both of these proposals, the size of the shares is dependent on the size of the secret.

By moving to the CSS realm, Krawczyk introduces a scheme with "short" shares with lengths independent of the secret's size [20]. A cryptographic key is shared using a PSS scheme, while the secret is encrypted using the key. The resulting ciphertext is shared using an information-dispersal algorithm, e.g., Rabin's IDA [27]. A share then consists of a cryptographic portion and a ciphertext portion. The cryptographic portion is at least as long as a cryptographic secret key plus a hash function image (thus, in practice, at least 384 bits). We use a similar mechanism to make the size of our shares independent of the secret, but in lieu of PSS and IDA schemes, we employ error correcting codes to reduce share sizes and add robustness.

We are aware of no investigation, however, of the particular problem of creating shares smaller than the short ones introduced by Krawczyk, i.e,, shares potentially *shorter* than a cryptographic secret key (perhaps 16 bits in length). Here, we characterize such shares as *tiny*.

The omission from the literature of CSS schemes with tiny shares appears to have two underlying causes. First, short shares are compact enough for many applications. Second, the literature is solidly anchored in PSS. Even CSS schemes, such as that of Krawczyk, typically rely on PSS as a primitive to share out cryptographic keys.

**Secret-sharing in RFID:** Langheinrich and Marti suggest using secret sharing to conceal an RFID tag's information from adversaries with time-limited access to the tag [21]. The tag's information is split using Shamir's scheme [32], and the tag periodically emits a share. A reader that probes the tag over the course of several minutes will receive enough shares to reconstruct the tag's information, while a casual attacker who only obtains a few emissions cannot reconstruct any tag information. Our schemes, in contrast, spread shares across multiple tags and consider sliding time windows with evolving secrets, rather than a single fixed secret.

In other work, Langheinrich and Marti propose using Shamir's scheme to distribute an item's ID over hundreds of RFID tags integrated into the item's material [22]. They aim to enforce privacy by requiring a reader to access multiple tags. In contrast, we look to dispersion, rather than aggregation, of tags, as a privacy-enforcing mechanism. We also reduce the size of each share to well below the size of standard Shamir shares.

## 5 Secret Sharing Across Space

Sharing a secret (e.g., a cryptographic key) across space in an RFID application imposes severe limitations on the size of each share. As discussed in Section 4, previous schemes typically require 128 bits or more for each share, whereas with RFID tags, we would like shares of 16 bits or less. Hence in this section we provide a generic robust secret sharing scheme that we refer to as a Tiny Secret Sharing (TSS) scheme. We define our scheme in a general problem framework based on adversarial games, describe a prototype implementation, and suggest parameters appropriate for real-world deployment.

### 5.1 Preliminaries

**Secret Sharing.** We adhere closely to the notation and definitions of Bellare and Rogaway [2]. An $n$-party *secret-sharing scheme* is a pair of algorithms $\Pi = (\mathsf{Share}, \mathsf{Recover})$ that operates over a message space $\mathbb{X}$, where:

- $\mathsf{Share}$ is a probabilistic algorithm that takes input $x \in \mathbb{X}$ and outputs the $n$-vector $S \xleftarrow{R} \mathsf{Share}(x)$, where $S_i \in \{0,1\}^*$. On invalid input $\hat{x} \notin \mathbb{X}$, $\mathsf{Share}$ outputs an $n$-vector of the special ("undefined") symbol $\perp$.
- $\mathsf{Recover}$ is a deterministic algorithm that takes input $S \in (\{0,1\}^* \bigcup \diamondsuit)^n$, where $\diamondsuit$ represents a share that has been erased (or is otherwise unavailable). The output $\mathsf{Recover}(S) \in \mathbb{X} \bigcup \perp$, where $\perp$ is a distinguished value indicating a recovery failure.

In our security definitions, we assume an honest dealer, i.e., correct execution of $\mathsf{Share}$ (although the adversary may choose the secret that is shared).

**Adversaries.** While secret sharing literature traditionally defines goals with respect to access structures, we predicate our definitions below on a class $\mathcal{A}$ of probabilistic adversarial algorithms. We define the security of a TSS scheme in terms of a particular class $\mathcal{A}$. We can reconcile our adversarial model with the traditional access-structure view by restricting $\mathcal{A}$ to only adversaries $A$ that respect a particular access structure. For example, we might consider only adversaries that compromise fewer than $d$ legitimate shares for some $d$.

**Error Correcting Codes.** Our construction utilizes an error-correcting code (ECC), a generalization of secret sharing that we formally define as a pair of algorithms $\Pi^{ecc} = (\mathsf{Share}^{ecc}, \mathsf{Recover}^{ecc})$. An $(N,K,D)_Q$-ECC operates over an alphabet $\Sigma$ of size $|\Sigma| = Q$. $\mathsf{Share}^{ecc}$ maps $\Sigma^K \rightarrow \Sigma^N$ such that the minimum Hamming distance in symbols between (valid) output vectors is $D$. For such a function $\mathsf{Share}^{ecc}$, there is a corresponding function $\mathsf{Recover}^{ecc}$ that recovers a message successfully given an attacker that can corrupt up to $D/2$ players or erase the shares of $D-1$ players—or some combination of the two, depending on the specific ECC. (In some cases, correction beyond the minimum distance is possible [28].)

### 5.2 Problem Definition

Informally, the adversary may attack either the privacy or the robustness of the scheme or both. A privacy attacker attempts to recover the secret $x$ given some number of shares. To break robustness, the adversary aims to corrupt shares such that $\mathsf{Recover}$ fails to output $x$. We define these security goals formally below and conclude with a definition of a TSS scheme.

#### 5.2.1 Privacy

We consider two subtypes of privacy attackers: an *underinformed* adversary and an *overinformed* adversary. An underinformed adversary can corrupt a limited number of players, while an overinformed adversary can obtain all $n$ shares, but also receives a number of additional "shares" that she cannot distinguish from the correct shares. Due to lack of space, we relegate details on overinformed adversaries to Appendix A. (Briefly, an overinformed adversary sees shares from multiple cases simultaneously, and cannot feasibly extract secrets due to the hardness of decoding given many "chaff" shares.)

**Underinformed Attacks.** Here, we consider an attacker who obtains a limited number of legitimate shares (recall Example 1). In this setting, Bellare and Rogaway define privacy based on a notion of indistinguishability. Given an $n$-party secret-sharing scheme $(\Pi, \mathbb{X})$, they define the oracle $\mathsf{corrupt}(S,i)$ as a function that returns $S_i$. ("Corruption" in this setting—corresponding to compromise of a share-holding player—results in disclosure, not change, of a share.) Then the Bellare and Rogaway notion of privacy is defined based on the experiment shown in Figure 2(a)

In the experiment, the adversary is asked to choose two values to be shared. The experiment selects one of the secrets at random and generates a set of shares. The adversary can then corrupt (or see the value of) individual shares and must eventually produce a guess as to which secret was shared. The corruptions and the guess may be based on state generated during the "choose" phase. Using this experiment, Bellare and Rogaway define $A$'s advantage as $\mathbf{Adv}^{ind}_A[\Pi,\mathbb{X}] \stackrel{\triangle}{=} 2\mathsf{Pr}\left[\mathbf{Exp}^{ind}_A[\Pi,\mathbb{X}] \Rightarrow 1\right] - 1$.

#### 5.2.2 Robustness

We desire our scheme to allow a legitimate user to recover the original secret, even if the adversary tampers with some of the shares. To model a scheme's resilience to such an attack, we define a robustness experiment. In

Experiment $\mathbf{Exp}^{ind}_{A}[\Pi,\mathbb{X}]$
$(x_0,x_1) \leftarrow A(\text{“choose”})$;
$b \xleftarrow{R} \{0,1\}$; $S \xleftarrow{R} \mathsf{Share}(x_b)$;
$b' \leftarrow A^{\mathsf{corrupt}(S,\cdot)}(\text{“corrupt”})$;
output ‘1’ if $b = b'$, else ‘0’

(a) **Privacy Experiment**

Experiment $\mathbf{Exp}^{rec}_{A}[\Pi,\mathbb{X}]$
$x \leftarrow A(\text{“choose”})$;
$S \xleftarrow{R} \mathsf{Share}(x)$;
$S' \leftarrow A^{\mathsf{corrupt}(S,\cdot)}(\text{“corrupt”})$;
$x' \leftarrow \mathsf{Recover}(\{S'_i\}_{i\in\hat{S}} \bigcup \{S_i\}_{i\notin\hat{S}})$;
output ‘1’ if $x \neq x'$, else ‘0’

(b) **Robustness Experiment**

Figure 2: **TSS Experiments.** *These experiments capture our notion of privacy and robustness for TSS schemes.*

our robustness experiment, Share is invoked on a secret $x$ of the adversary's choosing. The adversary then corrupts a number of players and *replaces their share values*. Again, the adversary is allowed to maintain state between the “choose” and the “corrupt” phases. The adversary is successful if Recover fails to recover $x$ given the corrupted and uncorrupted shares as input. This experiment is much like that for robustness in Bellare and Rogaway, though their definition additionally includes the technical requirement that the adversary identify an uncorrupted player $j$. This is not necessary for our purposes. We define the robustness experiment as shown in Figure 2(b), letting $\hat{S}$ represent the indices of the shares corrupted by the adversary. We define the advantage of $A$ as $\mathbf{Adv}^{rec}_{A}[\Pi,\mathbb{X}] \stackrel{\triangle}{=} \Pr[\mathbf{Exp}^{rec}_{A}[\Pi,\mathbb{X}] \Rightarrow 1]$.

It is also useful to consider a modified experiment $\mathbf{Exp}^{rec-or-detect}$ that outputs ‘1’ if $x \neq x'$ and $x' \neq \perp$, else ‘0.’ In other words, $A$ is successful if it causes a recovery failure that Recover *does not detect*. This is a weaker requirement, of course, than that represented by $\mathbf{Exp}^{rec}$, but an important condition not explored by Bellare and Rogaway. Given the above experiments, we define a TSS scheme as follows.

#### 5.2.3 TSS Definition

**Definition 1** *A $(k,n)$-TSS scheme is a pair $(\Pi,\mathbb{X})$, such that $\Pi$ distributes $n$ shares of a secret $x \in \mathbb{X}$, of which any set of $k$ correct shares suffices to recover $x$. The security of the scheme is characterized by an adversary class $\mathcal{A}$ and the tuple: $(q_u,\varepsilon_u,q_r,\varepsilon_r)$, where an underinformed attacker $A \in \mathcal{A}$ making $q_u$ corrupt queries has $\mathbf{Adv}^{ind}_{A}[\Pi,\mathbb{X}] \leq \varepsilon_u$; likewise, the pair $(q_r,\varepsilon_r)$ applies to robustness attackers. (An extended definition can include overinformed attackers as well; see Appendix A.)*

### 5.3 Our Construction

Figure 3 illustrates a high-level schematic of our TSS scheme. The $\mathsf{Share}^{TSS}$ algorithm accepts as input an arbitrarily-sized secret $x$. It then generates a large random pre-key $\tilde{\kappa}$. We apply a hash to reduce $\tilde{\kappa}$ to the size of a cryptographic key $\kappa$. The hash function also constitutes good cryptographic hygiene (and is used in our proofs) in the sense that it renders $\kappa$ indistinguishable even in the face of partial compromise of $\tilde{\kappa}$. We use the key $\kappa$ to perform authenticated encryption of $x$ and then use an $(N,K,D)$-error correcting code (ECC) to share both $\tilde{\kappa}$ and the ciphertext $\tilde{x}$. We focus in this paper on the basic construction that assigns a single symbol to each share. Thus we assume $K = k$. More general constructions are possible, but omitted from this paper. A recipient with enough shares can apply the ECC decoding algorithm to recover $\tilde{\kappa}$ and the ciphertext $\tilde{x}$, and then use $\tilde{\kappa}$ to derive the key $\kappa$ necessary to authenticate and decrypt $x$. In some applications (e.g., transporting the master key used to derive RFID kill codes), we may only want to distribute a key. In that case, we can use $\kappa$ as the desired key, and eliminate the portion of the schematic shown in the dashed box.

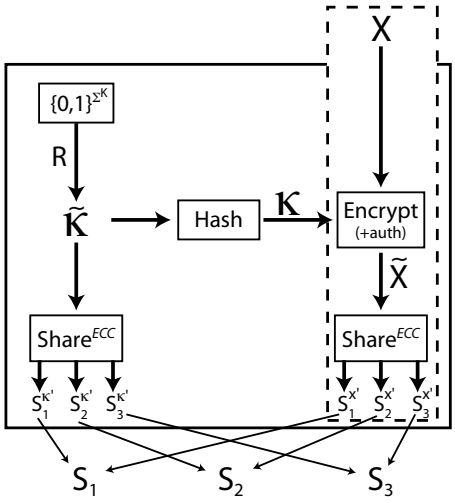



Figure 3: **Secret Sharing with Tiny Shares.** *Schematic of our TSS construction in a toy example with n=3. It can be used to distribute a key $\kappa$, or optionally a secret $x$ of arbitrary size. When $\kappa$ and $x$ are provided at the same time, the two error-correcting codes may be coalesced into a single one.*

Our construction assumes that the hash function behaves as a random oracle [1], and for large secrets, we assume the use of an authenticated encryption mode, such as OCB [29].

Below, we state our claims for the security of this construction. We defer the proofs to Appendix B.

**Claim 1** *Given our construction above, an underinformed attacker's advantage is bounded by* $\varepsilon_u$ *such that*

$$\mathbf{Adv}_A^{ind}[\Pi,\mathbb{X}] \leq \varepsilon_u \leq 1/Q^{k-q_u}.$$

**Claim 2** *Against an attacker who makes* $q_r$ *corrupt queries, if* $q_r < D/2$*, i.e.,* $q_r \leq \lfloor (D-1)/2 \rfloor$*, then* $\mathbf{Adv}_A^{rec}[\Pi,\mathbb{X}] = 0 = \varepsilon_r$*, and if* $q_r \leq D-1$*, then* $\mathbf{Adv}_A^{rec-or-detect}[\Pi,\mathbb{X}] = 0$.

Thus, our construction is a (k,n)-TSS scheme with security tuple $(q_u, 1/Q^{k-q_u}, \lfloor (D-1)/2 \rfloor, 0)$.

**Remark 1** *With an appropriate choice of an ECC, we can generalize the construction above. For example, using a systematic version of Reed-Solomon as the ECC,* $\tilde{\kappa}$ *will be encoded in the initial code symbols. We then apply a hash function (SHA-256 with truncation) to those code symbols to derive* $\kappa$*. If we choose* $Q = 2^{|\tilde{\kappa}|}$ *(and do not release* $S_1^{\tilde{\kappa}}$*), then* $Share^{ECC}$ *becomes a robust PSS scheme, as in Krawczyk's scheme [20]. If we choose* $Q = 2^n$*, then we have the scheme described above. Intermediate choices of Q trade increased share size for increased security.*

## 5.4 Implementation Sketch and Real World Parameterization

We implemented a $(15,20)$-TSS scheme using a ThingMagic Mercury5 reader and commercially-available Alien Squiggle Gen2 tags. A schematic view of the setup is shown in Figure 4. Use of a $(15,20)$-TSS scheme means that of the 20 available tags, we need to read at least 15 tags successfully to recover the key and decrypt tag data. We work over the field $GF(2^{16})$, so a share (codeword symbol) is 16 bits. The Share algorithm was then implemented as follows. We chose a secret key $\kappa$ of length 128-bits; we obtained $\kappa$ by choosing a random 240-bit value $\tilde{\kappa}$, hashing it with SHA-256, and then taking the first half of the output. We then encoded $\tilde{\kappa}$ into 20 16-bit symbols with a $(20,15)$ Reed-Solomon ECC using the built-in Reed-Solomon encoder in Matlab's Communication Toolbox. This resulted in 20 16-bit shares, one for each tag.

Given that we were using 96-bit tags, we had 80 bits left over for the tag ID. This particular parametrization requires a cipher with an 80-bit block size. We achieve this by using the Blowfish block cipher [31], which has a block size of 64 bits, with Ciphertext-Stealing [11] to expand the block size to 80 bits. Integrity protection at the individual tag ID level is provided by the Gen2 protocol.

Each tag ID $m_i, 1 \leq i \leq 20$, was then replaced by $E_\kappa[m_i]$ and concatenated with a share of $\tilde{\kappa}$ (as generated above). This combined 96-bit string was written into the tag using the same setup (Figure 4). Because all Gen2 RFID readers can also wirelessly write to tags, this operation is accomplished by bring each tag into the antenna field of the reader and executing a Gen2 write command. In practice, this operation would be carried out when the case, pallet, or item tag is initially encoded in the supply chain. Note that $E_\kappa$ as used here includes Ciphertext-Stealing as described above.

For the Recover algorithm, we simply unwound Share. As shown in Figure 4, the reader sees encrypted tag IDs with concatenated shares. As long as the reader sees more than 15 tags, Recover running on the PC outputs the tag IDs successfully.

In an ECC, a codeword consists of an *ordered* sequence of symbols. Because there is no fixed reading order for tags in our implementation, however, it must be *order invariant*. That is, since shares are not distributed among players with fixed identities, as in our robustness experiment, we must explicitly associate an index with each share (effectively assigning a player index to each tag). Thus, the symbol on a tag must be accompanied by an index specifying its codeword position. Rather than specifying this index explicitly, and thereby using an additional 16 bits of storage, we derive it implicitly based on the encrypted tag ID. In particular, we hash the ID using SHA-256, and interpret the last 16 bits as the index; of course, we must do this *before* sharing the encryption key. This optimization potentially introduces a new problem: Two (or more) tags within a case may have ciphertexts that hash to the same index. A sufficiently large index size can minimize this problem. (By the Birthday Paradox, $GF(2^{16})$ accommodates roughly 256 tags without many collisions.) As a further optimization, we can dedicate a few additional bits of storage to disambiguating collisions that do occur. Finally, if there are still too many collisions, we can simply choose a new random pre-key $\tilde{\kappa}$ and compute a new set of shares.

In general, the first step in parameterizing the TSS scheme for real-world usage is to determine the total number of tags $n$ and the key-recovery threshold $k$. As noted earlier (section 2.2), these numbers can vary widely between use cases. Today, pallets typically carry from 1 to 200 tags each. In a typical distribution center setting, an RFID reader could, depending on pallet composition, fail to read as many as 2–3% (i.e., 4–6) of the tags in a 200-item pallet, and it may pick up as many as 3–10 *stray* tags from a pallet in an adjacent dock door. This means that we can see up to 6 erasures, and up to 10 errors in reading. These numbers are borne out by one the authors' (RP) long experience in supply chain RFID deployments. Thus the choice of a (200, 170)-Reed Solomon code (the minimum distance $D = N - K + 1$ is typically omitted from Reed-Solomon parameterization), which can correct up to 15 errors or 30 erasures, would provide sufficient error correction for real-world deployments. As discussed in Section 2.2, individual consumers typically have fewer than 40 tags from the same case, so we could alternatively choose a (200, 40)-Reed Solomon code to maintain privacy and provide additional robustness to read errors.

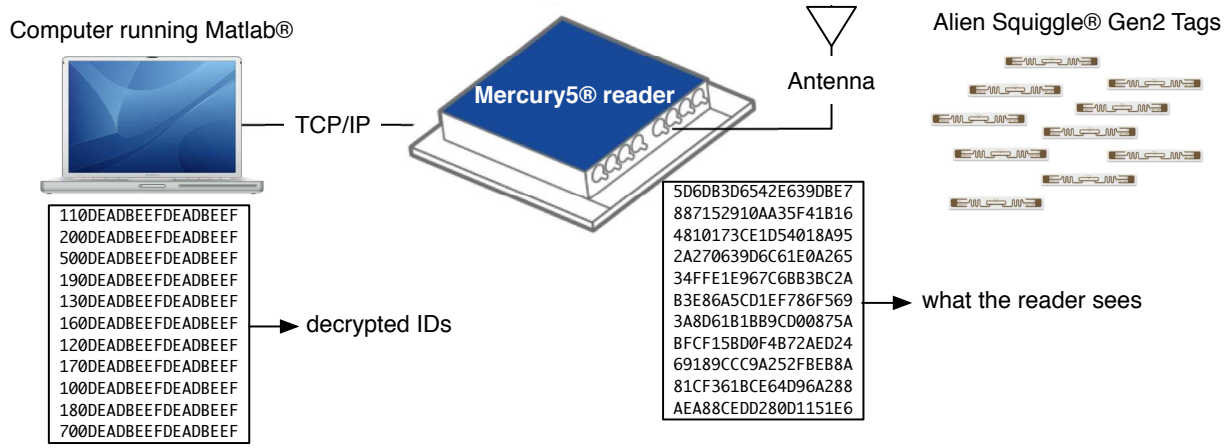



Figure 4: **Schematic of implementation setup** *20 TSS-encoded RFID tags, at far right, are prepared using* Share *as described in the text. They are read by a ThingMagic Mercury5 reader and the encrypted IDs are passed over the network to a Matlab program running* Recover *on a computer. The computer first recovers the Reed-Solomon-encoded secret key and then decrypts the tags. The two boxes below the schematic depict what the reader sees and the eventual decrypted tag IDs. In practice,* Recover *would be ported to run directly on the reader. Given the capabilities of current RFID readers, direct implementation on the reader is straightforward.*

Lastly, we remark on the choice of the field size. As the field size is the main determinant of the extra tag memory consumed by our scheme, smaller fields mean smaller memory requirements. Larger field sizes reduce the number of index collisions, which is useful both to ensure good decoding rates and to enforce security against an overinformed adversary (Appendix A). In applications where only the underinformed attacker must be considered, we can potentially reduce the space on each tag to a single bit, for sufficiently large $k$ and an appropriate ECC scheme.

## 6 Secret Sharing Across Time

Thus far, we have considered sharing schemes for one shipment. However, a distributor may wish to increase security by leveraging the fact that a legitimate recipient should receive more shipments than an attacker can access (recall Example 2 from Section 3). In this section, we explore a class of schemes that uses such information disparities across sliding time windows. In the future, we will investigate schemes leveraging the entropy of the entire history of interactions between a sender and recipient.

### 6.1 Defining SWISS: Sliding-Window Information Secret Sharing

In the schemes below, we assume a sender periodically emits a share $S_i$. For RFID purposes, we may suppose the sender is a manufacturer who periodically ships out containers of RFID-labeled items. Each share may optionally be further shared out amongst the RFID tags in the container as described in Section 5. Each period also has an associated key $\kappa_i$. Thus, we have a sequence of shares $S = \{S_0, S_1, \dots\}$ that expands indefinitely over time. We

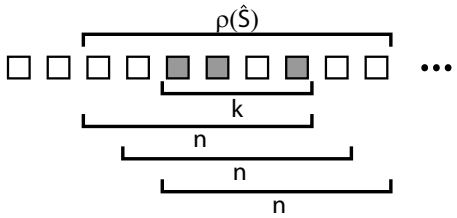



Figure 5: *In this example, if the adversary holds a set $\hat{S}$ of $k = 3$ shares (shown as shaded boxes), then we define $\rho(\hat{S})$ as the union of all (three) windows of $n = 6$ shares containing the original $k$ shares. We require that the adversary be unable to recover keys for periods outside of $\rho(\hat{S})$. The figure assumes $\lambda = 0$. If $\lambda = 1$, then $\rho(\hat{S})$ would include two additional shares: one before and one after the set $\rho(\hat{S})$ currently shown.*

assume that within any window of $n$ elements, only a legitimate recipient receives at least $k$ of the shares in that window, and given those shares, the recipient should be able to recover the corresponding keys. An adversary receiving fewer shares should learn nothing about the keys.

More formally, a SWISS scheme is defined as a pair of algorithms $\Pi = (\mathsf{Share}, \mathsf{Recover})$, where:

- $\mathsf{Share}(k, n, \tau)$ is a probabilistic algorithm that takes as input a threshold for recoverability $k$, a window size $n$, and a security parameter $\tau$. It outputs two "infinite" vectors $\kappa$ and $S$, where $\kappa_i \in \{0,1\}^\tau$ is the key for period $i$, and $S_i$ is the share for period $i$. On invalid input, Share outputs the special symbol $\perp$.
- Recover is a deterministic algorithm that takes as input $S' \subset W_j$ where $W_j$ defines a sequence of $n$ shares starting at time $j$ such that $W_j = \{S_i : j \le i < j+n\}$, and $|S'| \ge k$. The output of $\mathsf{Recover}(S')$ is a set of keys $K = \{\kappa_i : S_i \in S'\}$ for the shares provided in $S'$ or $\perp$, a special value indicating a recovery failure.

In our security definitions, we again assume an honest dealer, i.e., correct execution of Share. Below, we give formal definitions for our privacy and recoverability requirements.

**Privacy.** To define privacy, we require that the adversary cannot obtain the key for any share she does not possess. If the adversary holds fewer than $k$ shares, she should not learn any keys. We deal with the case in which the adversary holds more than $k$ shares as follows.

Define the set of shares held by the adversary as $\hat{S}$. Let $\rho(\hat{S})$ be the set of all shares that lie in a window of size $n+\lambda$ for which the adversary has recovered at least $k$ shares. We require the adversary to be unable to recover the key for any element in $\overline{\rho}(\hat{S})$, the complement of $\rho(\hat{S})$. Since $k$ shares allow the adversary to recover all of the keys in a window of size $n$, the value of $\lambda$ indicates the amount of information $k$ shares "leak" about keys not contained within a window of $n$ shares. Figure 5 illustrates these requirements.

More formally, we can define privacy based on the following experiment:

Experiment $\mathbf{Exp}_A^{ind-swiss}[\Pi]$
  $(S,\kappa) \stackrel{R}{\leftarrow} \mathsf{Share}(k,n,\tau)$;
  $i \leftarrow A$("choose");
  $\kappa^R \stackrel{R}{\leftarrow} \{0,1\}^{|\kappa|}$; $b \stackrel{R}{\leftarrow} \{0,1\}$;
  $b' \leftarrow A^{\mathsf{corrupt}(S,\cdot)}(\pi(b,\kappa^R,\kappa_i)$,"corrupt");
  if $i \notin \rho(\hat{S})$ or $i \notin \hat{S}$ then
    output '1' if $b' = b$, else '0';
  else output '0';

where $\pi(0,x,y) = (x,y)$ and $\pi(1,x,y) = (y,x)$. Essentially, the adversary is asked to choose a time period $i$. After corrupting some number of shares, the adversary must distinguish between the key for period $i$ and a randomly selected key. We consider the adversary successful if the period chosen does not correspond to a share held by the adversary, or if the period lies outside the set $\rho(\hat{S})$ induced by the adversary's selection of shares. The adversary's advantage is then $\mathbf{Adv}_A^{ind-swiss}[\Pi] \stackrel{\triangle}{=} 2\mathsf{Pr}\left[\mathbf{Exp}_A^{ind-swiss}[\Pi] \Rightarrow 1\right] - 1$.

**Recoverability.** We require that any set $S' \subseteq W_j$ with $|S'| \geq k$ shares suffices to recover the keys associated with each share in the set, namely $\{\kappa_i : S_i \in S'\}$. We define recoverability for a legitimate recipient in the erasure model; in other words, shares may be lost but not corrupted. We can convert our SWISS schemes to a corruption model by replacing our use of PSS schemes with robust PSS schemes, such as Krawczyk's [20].

**Definition 2** *We define a $(k,n)$-SWISS scheme as a pair of algorithms $\Pi$ as defined above where* Share *produces shares of size $\mu$. The security is characterized by the pair $(\lambda,\varepsilon)$, where (as explained above) $k$ shares are sufficient to reveal $\lambda$ "nearby" keys for time periods not contained in a window of $n$ shares, and $\mathbf{Adv}_A^{ind-swiss}[\Pi] \leq \varepsilon$.*

Thus, an ideal SWISS scheme would have $(\lambda,\varepsilon) = (0,0)$ with minimal $\mu$.

## 6.2 A Family of SWISS Schemes

In our SWISS construction, we want to ensure that the secret for a case is only available given possession of that case. To achieve this property, we make the key $\kappa_i$ for case $i$ a function of both a window key and a secret value associated with the case (or its RFID tag).

Ideally, the window key for a window of $n$ cases should be recoverable if and only if the receiver possesses $k$ or more cases within that window. A naïve SWISS scheme would simply generate a key for every possible window of size $n$ and share each key using a $(k,n)$ scheme. But a case would then need a share for every window covering it, and hence the per-case share size would grow linearly with the size ($n$) of each window.

Instead, we aim to bring the share size down to a small constant independent of $k$ and $n$. We use two techniques for this goal. First, we allow some sloppiness in our access structure. Our access structure (in our main construction) depends on superwindows of size $2n$ that each overlap with the previous superwindow by $n$ (see Figure 6); each superwindow secret is shared using a $(k,2n)$ scheme. Access to a window secret requires recovery of the secrets for either one of its two corresponding superwindows. Any $k$ shares in a sequence of size $n$ fall into some superwindow of size $2n$, and therefore allow recovery of the superwindow secret. The "sloppiness" is this: Recovery of shares in one window allows for recovery of secrets in nearby windows.

Given the superwindow scheme described above, we could encrypt the secret $\kappa_i$ for each case $i$ under each of its corresponding superwindow secrets, $\sigma$ and $\sigma'$. However, using a second technique based on bilinear maps, we can derive a common secret directly from either of the two superwindow secrets $\sigma$ *or* $\sigma'$.

Below, we first explain the assumptions necessary for our schemes. Then we present our main SWISS construction (Section 6.2.2) and show how to generalize it to a wider range of parameters (Section 6.2.3).

### 6.2.1 Assumptions

Our family of SWISS schemes uses bilinear pairing to reduce storage costs. In the full version of this paper, we describe a variant of our SWISS construction based on the more standard RSA assumption. Unfortunately, that version does not generalize efficiently to large window sizes in the same way as does the bilinear map scheme, and hence we focus on the latter.

We give some very brief preliminaries on bilinear maps, referring the reader to [7] for details. Let $E$ be a multiplicative cyclic group of prime order $p$ under a bilinear operator $\hat{e}$ as defined in Boneh-Franklin [7]. Thus we have $\hat{e}: E \times E \rightarrow E'$ for a second group $E'$. The bilinear operator $\hat{e}$ has the property that $\forall G, H \in E, \hat{e}(G^a, H^b) = \hat{e}(G,H)^{ab}$; it is also non-degenerate, meaning that if $G$ is a generator of $E$, then $\hat{e}(G,G) \neq 1$.

Our work relies on the hardness of a slightly modified Bilinear Diffie-Hellman Exponent (BDHE) problem [6,8]. Specifically, let $g$ and $\gamma$ be random generators of $E$, and $\alpha$ be a random element in $\mathbb{Z}_p^*$. Our $(\ell,L)$-BDHE problem is defined as:

$$\begin{aligned} &\text{Given } g, \gamma, g^{(\alpha^i)} \quad \text{for } i = 1,2,...,\ell-L,\ell+1,...,2\ell \\ &\qquad \text{and } \gamma^{(\alpha^i)} \quad \text{for } i = 1,2,...,L-1 \\ &\text{compute } \hat{e}(g,\gamma)^{(\alpha^\ell)}. \end{aligned}$$

In the original framing of the $\ell$-BDHE problem [6, 8], only $\gamma$ (rather than additional $\alpha$ powers of $\gamma$) is assumed to be known. We assume that $L \geq 2$, since the $(\ell,1)$-BDHE problem simply degenerates to the $\ell$-BDHE problem. Loosely speaking, the $(\ell,L)$-BDHE assumption in $E$ says that no efficient algorithm can solve the $(\ell,L)$-BDHE problem in $E$ with non-negligible probability.

We can apply the "master" theorem of Boneh et al. [6] to bound the difficulty of $(\ell,L)$-BDHE in a generic group. In their terminology, we have $P = (1, y, y^2, ..., y^{L-1}, x, x^2, ..., x^{\ell-L}, x^{\ell+1}, ..., x^{2\ell})$, $Q = (1)$ and $f = x^\ell y$. This implies that an attacker $A$ with advantage $1/2$ in solving the decision $(\ell,L)$-BDHE problem in a generic bilinear group $E$ must take time at least $\Omega\left(\sqrt{p/(4\ell)} - 2\ell\right)$. E.g., if we assume the distributor sends one billion windows (or less), then solving the decision $(\ell,L)$-BDHE problem in a generic bilinear group $E$ of size 192 bits takes time at least $2^{80}$. Of course, a lower bound in a generic group does not imply a lower bound in any specific group.

### 6.2.2 Our Main SWISS Construction

In Section 6.2.3, we present a fully generic overlapping SWISS scheme, but first, to simplify the exposition, we describe a single member of the family (see Figure 6). This example provides a $(k,n)$-SWISS scheme with $\mu = 3\tau$ and security parameters $(2n-k, \varepsilon)$.

Starting at time 0, the sender defines a series of superwindows $W_0, W_n, W_{2n}, ..., W_{\ell n}$, each of size $2n$. Thus, each superwindow consists of two windows of size $n$, with one window overlapping a window from the previous superwindow, and one window overlapping a window from the subsequent superwindow. Each superwindow $W_{\ell n}$ defines a $(k,2n)$ perfect secret sharing (PSS) of the superwindow secret $\sigma_{\ell n}$. Since each time period $i$ is covered by two superwindows, each comprising its own secret sharing scheme, the share $S_i$ distributed in each time period

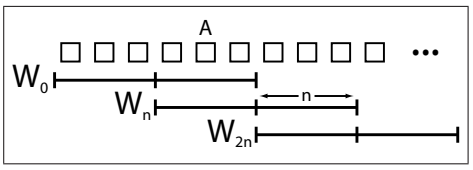



Figure 6: *Each superwindow of $2n$ shares (in the example shown here, $n = 3$) overlaps with the previous superwindow by n shares. Each superwindow $W_{\ell n}$ is a $(k,2n)$ sharing of the superwindow secret $\sigma_{\ell n}$. Each time period is covered by two superwindows. For example, the share labeled A is covered by superwindows $W_0$ and $W_n$. As a result the key for that period $\kappa_A$ can be recovered from either superwindow secret, $\sigma_0$ or $\sigma_n$.*

consists of two sub-shares $(s_i^{\ell n}, s_i^{(\ell+1)n})$, one for $\sigma_{\ell n}$ and one for $\sigma_{(\ell+1)n}$. We also augment the share with a random nonce $r_i \overset{R}{\leftarrow} \{0,1\}^\tau$. Thus, the share emitted during time period $i$ is $S_i = (s_i^{\ell n}, s_i^{(\ell+1)n}, r_i)$.

Because any time period $i$ is covered by two superwindows (say $W_{\ell n}$ and $W_{(\ell+1)n}$), we would like the key $\kappa_i$ to be recoverable from the superwindow secret of either one (since we do not know a priori in which superwindow the recipient will have $k$ shares). Like many problems in computer science, we can solve this by adding another layer of indirection. Let $y, z \in E$, $a \in \mathbb{Z}_p^*$, and let $(P_0, P_1) = (y, y^a)$ be a public key. Let each of the superwindow secrets be defined so that $\sigma_{\ell n} = z^{a^\ell}$. We define a series of window secrets $\omega_0, \omega_n, ..., \omega_{\ell n}$ so that

$$\omega_{\ell n} = \hat{e}(P_1, \sigma_{\ell n}) = \hat{e}(P_0, \sigma_{(\ell+1)n}) = \hat{e}(y,z)^{a^{\ell+1}}.$$

That is, knowing $\sigma_{\ell n}$ allows a recipient to derive $\omega_{\ell n}$ and $\omega_{(\ell+1)n}$.

Finally, we define each key $\kappa_i$ based on the window it belongs to, as well as the random nonce $r_i$ distributed with share $S_i$, as $\kappa_i = h(r_i, \omega_{kn})$, where $h: \{0,1\}^* \rightarrow \{0,1\}^\tau$ is a hash function modeled as a random oracle [1].

In the next section, we show how to generalize this construction to decrease $\lambda$ at the cost of increasing the size of each share. We can define an adversary for this more general scheme as follows:

**Definition 3** *We define an $(\ell,L,q)$-adversary A as an attacker who achieves an $\mathbf{Adv}_A^{ind-swiss}[\Pi] < \varepsilon$ advantage in our privacy experiment (defined in Section 6.1), where $\Pi$ is an instantiation of our generic SWISS family with $\Psi = L-1$ (for $L \geq 2$) that produces at most $2\ell$ shares. The adversary makes at most q random oracle queries.*

In Appendix C, we use this definition to demonstrate the security of the generalized scheme (and hence this specific instantiation) by proving the following theorem:

**Theorem 1** *For any polynomial-time $(\ell,L,q)$-adversary A with $\mathbf{Adv}_A^{ind-swiss} = \varepsilon$ and $\ell > L \geq 2$, there is a polynomial-time adversary $A'$ that solves the $(\ell,L)$-BDHE problem with probability $(\varepsilon - 2^{-\tau})/q\ell - 1/2^\tau$.*

Essentially the theorem states that given an adversary who achieves a non-negligible advantage in our privacy experiment, we can construct an attacker who violates the $(\ell, L)$-BDHE assumption. We also demonstrate that this construction satisfies our recoverability requirement.

**Remark 2** *As described, our SWISS construction uses a PSS scheme to create superwindow shares. Thus, the construction tolerates erasures but not errors. However, we could readily replace the PSS scheme with a robust scheme, such as our TSS scheme from Section 5, which would both decrease the size of the individual shares and add error tolerance to the SWISS construction.*

#### 6.2.3 A Generic SWISS Family

The above scheme can be generalized to allow decreased values of $\lambda$ at the cost of increased storage (see Figure 7). Specifically, for any value of $\Psi < n$, we can create a $(k, n)$ SWISS scheme with $\mu = (\Psi + 2)\tau$ and security parameters $((1 + \frac{1}{\Psi})n - k, \varepsilon)$.

Essentially, we divide each superwindow $W$ into $\Psi + 1$ windows of size $\frac{n}{\Psi}$. The superwindows form $(k, \frac{(\Psi+1)n}{\Psi})$ sharing schemes of the superwindow secrets, and each superwindow overlaps the previous superwindow by $\Psi$ windows. Thus, any given window is covered by $\Psi + 1$ superwindows, and the window secret can be recovered from any of the superwindow secrets, using the same elliptic curve pairings technique as before. In other words, we define a public key $(P_0, P_1, ..., P_\Psi) = (x, x^a, ..., x^{a^\Psi})$, and a window secret $\omega_{\ell n}$ is defined as:

$$\begin{aligned} \omega_{\ell n} &= \hat{e}(P_\Psi, \sigma_{\ell n}) = \hat{e}(P_{\Psi-1}, \sigma_{(\ell+1)n}) = ... \\ &= \hat{e}(P_0, \sigma_{(\ell+\Psi)n}) = \hat{e}(x, z)^{\ell+\Psi}. \end{aligned}$$

To determine $\lambda$, we consider the worst case, in which $k \leq \frac{n}{\Psi}$, and the adversary's $k$ shares fall within a single window. The window then is covered by $\Psi + 1$ superwindows, allowing the adversary to recover secrets for $2\Psi + 1$ windows, or $(2\Psi + 1)\frac{n}{\Psi} = 2n + \frac{n}{\Psi}$ secrets. These secrets can be at most a superwindow $(\frac{\Psi+1}{\Psi}n)$ away from the $k$ secrets held by the adversary, so $\lambda = \frac{\Psi+1}{\Psi}n - k = (1 + \frac{1}{\Psi})n - k$. If $k > \frac{n}{\Psi}$, then fewer than $\Psi + 1$ superwindows will contain $k$ shares, and hence $\lambda$ will be even smaller.

In our example scheme from Section 6.2.2, $\Psi = 1$, so each superwindow formed a $(k, 2n)$ secret-sharing scheme, but we could also use $\Psi = 2$, with each superwindow consisting of 3 windows of $\frac{n}{2}$ shares, and the superwindow as a whole constituting a $(k, \frac{3}{2}n)$ sharing of the superwindow secret (see Figure 7(a)). This would produce a smaller value of $\lambda = \frac{3}{2}n - k$, but at the cost of larger shares: each issued share would now contain three shares (one for each superwindow overlapping a particular window) and the random nonce $r_i$.

#### 6.2.4 Real World Instantiation

To make our SWISS construction more concrete, we suggest sample parameters for real world deployments. Suppose the sender needs to ship one million or fewer shares. We divide those shares into 10,000 windows of 100 shares each, giving us $\ell = 5,000, n = 100$. A legitimate recipient will receive at least $k = 20$ shares in any window. If we use the scheme from Section 6.2.2, then $\Psi = 1$ and $L = \Psi + 1 = 2$. Finally, if we use $\tau = 128$ bit keys, then the share for each period will be $3\tau = 384$ bits in size. In contrast, the naïve scheme described earier in this section would require $n\tau = 12,800$ bits per share.

We described both our SWISS scheme and the naïve scheme using PSS as a component. If we replace the PSS scheme with our TSS scheme from Section 5, then we have a share size of 16 bits. In our scheme, we still need a random nonce of at least 60 bits, but that yields shares of size $2 \cdot 16 + 60 = 92$ bits, just small enough to fit on an EPC tag. In contrast, the naïve scheme would require $n \cdot 16 = 1,600$ bits.

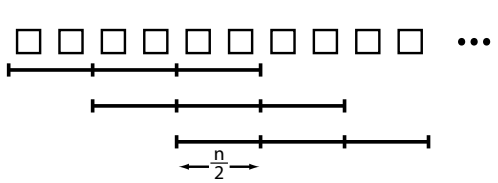
## 7 Conclusions and Future Work

We have described two approaches to secret sharing in unidirectional channels: secret-sharing across space and secret-sharing across time. As we have shown, secret-sharing across space is a tool of practical promise for privacy protection in real-world RFID-enabled supply chains. Our SWISS scheme for secret-sharing across time can, similarly, help address the challenges of RFID tag and reader authentication. An open problem of particular interest in our SWISS construction, however, is elimination of its reliance the non-standard $(\ell, L)$-BDHE problem in our fully generic overlapping SWISS scheme. We also plan to investigate extended SWISS schemes that leverage the entire history of interaction between a sender and receiver, rather than simply a window of recent history.

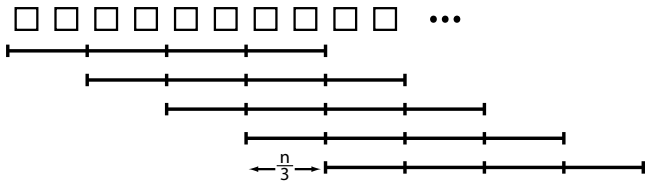
More broadly, we believe that a holistic view of the special operational requirements of RFID tags and the highly constrained resources of tags can give rise to important new cryptographic problems. Our future work will aim to calibrate cryptographic tools like those presented here to RFID supply-chain infrastructure as it evolves and its special operational demands come into clearer focus.

## 8 Acknowledgements

The authors would like to thank Burt Kaliski, Jonathan McCune, Alina Oprea, and Diana Parno for their helpful feedback and editorial suggestions. We are also grateful for the comments from the anonymous reviewers.




(a) **A SWISS scheme with $\Psi = 2, n = 4$.** Each superwindow is a $(k, 3n/2)$ sharing of the superwindow secret.




(b) **A SWISS scheme with $\Psi = 3, n = 6$.** Each superwindow shown is a $(k, 4n/3)$ sharing of the superwindow secret.

Figure 7: **Additional SWISS examples** *We can create additional SWISS schemes by increasing the number of windows per superwindow while decreasing the number of shares in each window. As we increase the number of windows, $\lambda$ decreases, but the number of shares that must be held in each time period increases.*

## References


[1] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.

[2] M. Bellare and P. Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *ACM CCS*, 2007.

[3] C. H. Bennett, G. Brassard, C. Crepeau, and U. Maurer. Generalized privacy amplification. In *ISIT: Proceedings IEEE International Symposium on Information Theory*, 1994.

[4] G. Blakley. Safeguarding cryptographic keys. In *AFIPS Conference Proceedings*, volume 48, pages 313–317, 1979.

[5] G. Blakley and C. Meadows. Security of ramp schemes. In *Advances in Cryptology: Proceedings of CRYPTO*, 1984.

[6] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456, 2005.

[7] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[8] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. *Advances in Cryptology: Proceedings of CRYPTO*, 2005.

[9] E. F. Brickell and D. R. Stinson. Some improved bounds on the information rate of perfect secret sharing schemes. *Journal of Cryptology*, 5:153–166, 1992.

[10] R. M. Capocelli, A. D. Santis, L. Gargano, and U. Vaccaro. On the size of shares for secret sharing schemes. *Journal of Cryptology*, 6:157–167, 1993.

[11] J. Daemen. *Hash Function and Cipher Design: Strategies Based on Linear and Differential Cryptanalysis*. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Mar. 1995.

[12] EPC Global. EPC® Radio-Frequency Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz Version 1.1.0. *EPC Global*, 2006.

[13] EPC Global. EPC® Item Level Tagging Joint Requirements Group. *EPC Global*, 2007.

[14] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[15] A. Juels. Strengthing EPC tags against cloning. In *ACM Workshop on Wireless Security (WiSe)*, pages 67–76. ACM Press, 2005.

[16] A. Juels, D. Molnar, and D. Wagner. Security issues in e-passports. In *SecureComm*, 2005.

[17] A. Juels and M. Sudan. A fuzzy vault scheme. *Des. Codes Cryptography*, 38(2):237–257, 2006.

[18] E. D. Karnin, J. W. Greene, and M. E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29(1):35–41, 1983.

[19] A. Kiayias and M. Yung. Directions in polynomial reconstruction based cryptography. *IEICE Transactions*, E87-A(5):978–985, 2004.

[20] H. Krawczyk. Secret sharing made short. In *Advances in Cryptology: Proceedings of CRYPTO*, pages 136–146, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[21] M. Langheinrich and R. Marti. Practical minimalist cryptography for RFID privacy. In submission, 2007.

[22] M. Langheinrich and R. Marti. RFID privacy using spatially distributed shared secrets. In *Proceedings of UCS 2007*, LNCS, Berlin Heidelberg New York, Nov. 2007. Springer. (To appear).

[23] J. L. Massey. Shift register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[24] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-Believing: Using camera phones for human-verifiable authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2005.

[25] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[26] W. Ogata and K. Kurosawa. Some basic properties of general nonperfect secret sharing schemes. *Journal of Universal Computer Science*, 4(8), 1998.

[27] M. O. Rabin. The information dispersal algorithm and its applications, 1990.

[28] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal SIAM*, 8:300–304, 1960.

[29] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM TISSEC*, Nov. 2001.

[30] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *ACM Workshop on Wireless Security (WiSe 2003)*, pages 1–10, Sept. 2003.

[31] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In R. J. Anderson, editor, *FSE*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 1993.

[32] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[33] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols, 7th International Workshop*. Springer Verlag, 1999.

## A Overinformed Adversaries

In the body of the paper, we discuss the notion of an underinformed adversary, one that has an insufficient set of shares to reconstruct a secret key. We also briefly consider an *overinformed* adversary., one that possesses a set of shares sufficient to reconstruct one or more secret keys, but has too many shares to feasibly determine such keys. We can design our system such that an adversary is overinformed in settings where the adversary is forced to scan the contents of not one, but *multiple* cases simultaneously.

Consider, for example, an attacker who periodically scans a store shelf, hoping to accumulate enough shares to recover the associated key. The adversary's reader may receive responses from items that arrived in multiple independent cases. In this situation, we would like it to be hard for the adversary to recover any case secret from the full set of secrets, even if a subset of the adversary's shares would suffice to reconstruct the secret. We can appeal to the fact that when shares from multiple cases are mixed together, the large set of shares can make it hard to decode any individual secret.

To help render an attacker overinformed, we can deliberately introduce "chaff" among the shares $S_i$ in a case. Essentially, we replace $\zeta$ shares of $\tilde{\kappa}$ with randomly chosen values. The choice of $0 \leq \zeta < D/2$ represents a tradeoff between security against an overinformed attacker and the error-tolerance of the scheme. For example, by choosing $\zeta = \frac{D}{3}$, an adversary who recovers the shares from two secrets will hold $\frac{2D}{3}$ chaff values—potentially exceeding the recovery threshold for the ECC scheme, as we now show. In this situation, however, a legitimate recipient can still tolerate $\frac{D}{6}$ errors in the shares she receives.

The following experiment formalizes the notion of an overinformed adversary.

Experiment $\mathbf{Exp}_A^{ind'}[\Pi, \mathbb{X}, \alpha, \beta]$
$(x_1, ..., x_\alpha) \xleftarrow{R} \mathbb{X}$;
$C \xleftarrow{R} \bigcup_{i=1}^{z} C^i$, where $C^i \overset{R}{\subseteq} \mathsf{Share}(x_i)$, and $|C^i| = \beta$;
$H \leftarrow \{h : h = \mathbb{H}(x_i), 1 \leq i < \alpha\}$;
$x' \leftarrow A^{\mathsf{corrupt}(C,\cdot)}(H, \text{"corrupt"})$;
output '1' if $x \in (x_1, ..., x_\alpha)$, else '0'

In this experiment, we choose $\alpha$ random secrets. The adversary has access to an unlabeled set of shares, which contains $\beta$ randomly chosen shares from each secret. The adversary also receives the hash $\mathbb{H}$ of each secret. Given this information, the adversary must recover one of the original secrets. In this experiment, we define the advantage of adversary $A$ as $\mathbf{Adv}_A^{ind'}[\Pi, \mathbb{X}, \alpha, \beta] \overset{\triangle}{=} \mathsf{Pr}\left[\mathbf{Exp}_A^{ind'}[\Pi, \mathbb{X}, \alpha, \beta] \Rightarrow 1\right]$.

We can characterize the overinformed adversary's task in terms of the *polynomial reconstruction (PR)* problem, the decoding of a Reed-Solomon codeword in the presence of errors (see [19] for detailed discussion).

Given an underlying $(N, K)$-Reed-Solmon code, and a set of $t$ symbols, of which $\zeta$ are corrupted, the classical Peterson-Berlekamp-Massey (PBM) algorithm [23] successfully decodes a set of symbols if $t - \zeta \geq (t + K)/2$ (or, equivalently, $\zeta \leq (t - K)/2$. A more powerful decoding scheme is that of Guruswami and Sudan (GS) [14], which successfully decodes for $t - \zeta > \sqrt{KN}$ in any field of cardinality at most $2^N$. It is conjectured that decoding beyond the error bound represented by GS is infeasible in a general sense and thus that GS offers a likely bound on the hardness of the PR problem.

That said, there are different formulations of the PR problem and little work on the concrete hardness of the problem. Schemes that achieve unconditional security, e.g., [17] do not offer attractive parameterization ranges for our purposes. Choosing credible and practical hardness assumptions for an overinformed adversary in our scheme is an open problem.

## A.1 Parameterization of Our RFID Secret-Sharing Scheme

We give a brief characterization of what we believe to be secure and feasible parameterizations of our scheme. These parameterizations permit PBM decoding for the legitimate reading of a single RFID-tagged case and at the same time exceed the GS bound for security against over-informed adversaries. We emphasize, however, that further research is needed for firm determination of the security of our scheme in a concrete sense.

Suppose that a case contains $N$ tags, of which $\zeta$ are chaff. PBM decoding for a scanned case is always possible when the number of corruptions (or erasures) of valid symbols $e$ is such that $N-(e+\zeta) \geq (N+K)/2$.

**Example 3** *Suppose that $K = 8$, $N = 200$, and $\zeta = 86$. Then it is possible to recover the secret associated with a case for $e \leq 10$, and thus up to a 5% corruption of tag symbols.*

Suppose that an adversary reads symbols associated with $q$ cases and attempts to recover the secret $x$ associated with a particular case. We can establish a lower bound on the hardness of this problem by rendering the problem easier for the adversary. In particular, let us assume that the adversary has access to an oracle that identifies valid shares associated with the $q-1$ untargeted cases (but does not otherwise reveal which shares correspond to which case). Then the adversary can reduce the problem of recovering $x$ to a decoding problem with $N-\zeta$ valid shares and $\zeta q$ chaff shares, and thus $t = N+(q-1)\zeta$ shares in total. The GS bound implies that recovery of $x$ is hard if $N-\zeta < \sqrt{K(N+(q-1)\zeta)}$.

**Example 4** *Suppose that $K = 8$, $N = 200$, and $\zeta = 86$. Then the problem of recovering a target case secret $x$ is hard under the GS bound if $114 < \sqrt{848+688q}$, and thus for $q \geq 18$.*

A stronger bound is possible assuming that valid symbols, i.e., secret-bearing data, in untargeted cases may be regarded as chaff. This gives us a slightly unorthodox problem distribution in which a problem instance has $q$ embedded, secret polynomials. In this case, however, the GS bound implies that recovery of $x$ is hard if $N-\zeta < \sqrt{qKN}$. With an appropriate parameter choice, we can obtain strong concrete results.

**Example 5** *Suppose that $K = 100$, $N = 200$, and $\zeta = 40$ (giving a 5% correction buffer in the single-case setting, as above). Then the problem of recovering a target case secret $x$ is hard under the GS bound if $160 < \sqrt{20000q}$, and thus for $q \geq 2$.*

# B Proofs of Security for Our Tiny Secret Sharing (TSS) Scheme

## B.1 Proof of Privacy

Since many of our applications only require the distribution of a secret key, we first define a simplified experiment to measure the indistinguishability of $\kappa$. Note that for this experiment, we excise the portion of our scheme in the dotted box in Figure 3. Effectively, we share out a null secret $x$, and write $\mathsf{Share}()$ to indicate this fact. The proof of privacy for secrets of arbitrary size then follows in a straightforward manner.

We define a key indistinguishability experiment as:

Experiment $\mathbf{Exp}_A^{ind-\kappa}[\Pi,\mathbb{X}]$
$(\kappa^0,S^0) \stackrel{R}{\leftarrow} \mathsf{Share}()$; $(\kappa^1,S^1) \stackrel{R}{\leftarrow} \mathsf{Share}()$;
$b \stackrel{R}{\leftarrow} \{0,1\}$;
$b' \stackrel{R}{\leftarrow} A^{\mathsf{corrupt}(S^b,\cdot)}(\kappa^0,\kappa^1,\text{“corrupt”})$;
output ‘1’ if $b=b'$, else ‘0’

In this experiment, the adversary receives two secret keys generated by our sharing algorithm, as well as the shares corresponding to one of the keys and must determine to which key they correspond. We define the advantage of adversary $A$ as $\mathbf{Adv}_A^{ind-\kappa}[\Pi,\mathbb{X}] \stackrel{\triangle}{=} 2\Pr\left[\mathbf{Exp}_A^{ind-\kappa}[\Pi,\mathbb{X}] \Rightarrow 1\right] - 1$.

For a generic ECC, if the adversary makes at most $q_u$ corrupt queries, then her total amount of information is upper-bounded by $Q^{q_u}$. Since we model the hash function applied to pre-key $\tilde{\kappa}$ as a random oracle, the adversary's advantage in distinguishing $\kappa^0$ and $\kappa^1$ is bounded above by $\mathbf{Adv}_A^{ind-\kappa}[\Pi,\mathbb{X}] \leq 1/Q^{k-q_u}$. Assuming an encryption algorithm in which key indistinguishability implies ciphertext indistinguishability (e.g., in an ideal cipher model), this bound then translates to the more general sharing of an arbitrary secret. Thus, we have $\mathbf{Adv}_A^{ind}[\Pi,\mathbb{X}] \leq \varepsilon_u \leq 1/Q^{k-q_u}$. This yields Claim 1 from Section 5.3.

## B.2 Proof of Robustness

With a generic linear $(N,K,D)$-ECC, it is possible to recover a message from a codeword with fewer than $D/2$ errors. Thus, as long as the adversary does not corrupt $D/2$ shares, $\varepsilon_r = 0$. Similarly, such a code can recover from $D-1$ erasures; and can also detect up to $D-1$ errors. As discussed in Appendix A, we can deliberately introduce $\zeta$ chaff shares into the ECC to confound the overinformed adversary. This would change are security parameters such that if $q_r < D/2-\zeta$, then $\mathbf{Adv}_A^{rec}[\Pi,\mathbb{X}] = 0 = \varepsilon_r$, and if $q_r \leq D-1-\zeta$, then $\mathbf{Adv}_A^{rec-or-detect}[\Pi,\mathbb{X}] = 0$. This yields Claim 2 from Section 5.3.

## C Proofs of Security and Recoverability for our SWISS Scheme

We prove that our generic family of SWISS schemes from Section 6.2.3 meets our privacy and recoverability requirements. Since our main construction from Section 6.2.2 is a specific instantiation (with $\Psi = 1$), its security follows from the security of the generic family of schemes.

### C.1 Proof of Privacy

To demonstrate that our generic family of SWISS schemes achieves our privacy requirement, we prove Theorem 1 based on the adversary specified in Def. 3. Recall that our generic family of SWISS schemes is parameterized by $\Psi$, one less than the number of overlapping superwindows.

**Proof of Theorem 1:** Suppose we are given an $(\ell, L)$-BDHE instance comprising $\gamma^{(\alpha^i)}$ for $i = 1, 2, ..., L-1$ and the sequence $U' = g'^{(\alpha^i)}$ for $i = 1, 2, ..., \ell - L, \ell + 1, ..., 2\ell$. We construct a SWISS-scheme simulator based on an $(\ell, L, q)$-adversary $A$ as follows.

**Simulator Construction.** First, we construct an appropriate public key by letting $(P_0, P_1, ..., P_{L-1}) = (\gamma, \gamma^{\alpha}, ..., \gamma^{\alpha^{L-1}})$. Then, we select a random $j \in \{1, ..., \ell\}$. This index is our guess as to the superwindow in which the adversary will select a challenge key. If we let $g = g'^{(\alpha^{\ell - j})}$, then $U'$ contains the subsequence $U = g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^{j-L}}, g^{\alpha^{j+1}}, ..., g^{\alpha^{\ell}}$.

We use this subsequence $U$ as the set of underlying superwindow keys in the procedure described in Section 6.2.2, with each superwindow representing a $(k, \frac{\Psi+1}{\Psi}n)$ sharing of $g^{(\alpha^i)}$. For the superwindows corresponding to $g^{(\alpha^{j-L+1})}, ..., g^{(\alpha^j)}$ (which are unknown), we simply share a random value. This procedure creates a set $S$ of shares. If $A$ queries $corrupt(S, i)$, we respond with $S_i$.

To respond to hash queries, we keep a list $\mathcal{V}$ of previous queries. Thus, when $A$ invokes $h(y, z)$ for the first time, we choose a random value $v \stackrel{R}{\leftarrow} \{0,1\}^{\tau}$ and add $(y, z, v)$ to the internal list $\mathcal{V}$. If $A$ has previously invoked $h$ on $(y, z)$, then we return the corresponding value of $v$ from $\mathcal{V}$. This creates a perfect implementation of the random oracle contract.

When $A$ terminates, we ignore its output, choose a random hash response $(y, z, v) \stackrel{R}{\leftarrow} \mathcal{V}$ and return $z$.

**Simulator Correctness.** From the SWISS adversary's point of view, the construction above accurately simulates the *ind-swiss* Experiment. Our replies to the hash queries perfectly instantiate a random oracle, so they offer the adversary no information with which to distinguish a real experiment from a simulation. Our construction deviates from the true protocol in one important respect: the keys for the superwindows corresponding to $g^{(\alpha^{j-L+1})}, ..., g^{(\alpha^j)}$ are chosen at random (since we do not know the appropriate values). However, the definition of $\rho$ precludes the adversary from recovering these superwindow secrets, and hence, she cannot determine that these values do not conform to the expected structure. Nonetheless, because we choose the superwindow secrets at random, we cannot provide the adversary with the correct value of $\kappa_i$. In other words, from our perspective, the value of $\kappa_i$ provided to the adversary is a random value. At some point, the adversary will query $h(r_i, \omega_{kn})$, but since we cannot recognize $\omega_{kn}$, we will not know that we should return $\kappa_i$. Fortunately, by the time the adversary makes this query, we have already extracted the necessary information, namely $\omega_{kn}$, so that even if the adversary quits upon determining a discrepancy, we will still succeed.

**Probability of Success** Our guess $j$ for the superwindow from which $A$ selects a challenge key $\kappa_i$ is correct with probability $\geq 1/\ell$. Since $h$ has a range of $\{0,1\}^{\tau}$ and $A$ has an $\varepsilon$ advantage, it is clear under the random oracle assumption on $h$ that $A$ inputs $\omega_{jn}$ with probability $\geq \varepsilon - 2^{-\tau}$. If $A$ has queried $h$ with $\omega_{jn}$ in the course of the simulation, then the probability that we output the correct $\omega_{jn} = \hat{e}(g, \gamma)^{(\alpha^{\ell})}$ is just $1/q$.

The only other way the adversary can succeed is by recovering a key for a share she does not hold. However, without the share, the adversary has no knowledge of $r_i$. The random oracle assumption on $h$ guarantees that the adversary succeeds in guessing $\kappa_i$ with probability less than $1/2^{\tau}$. Our theorem bound follows. ■

### C.2 Proof of Recoverability

A legitimate receiver (one who recovers at least $k$ shares out of some window $W'$ of $n$ shares) can determine the key corresponding to each share. Observe that given the overlapping superwindow construction, the window $W'$ is entirely contained within at least one superwindow $W_{\ell n}$. Thus, $k$ elements from $W'$ suffice to reconstruct the superwindow secret $\sigma_{\ell n}$, which can be used to calculate the window secrets $\omega_{\ell n}$, $\omega_{(\ell+1)n}$, ..., $\omega_{(\ell+\Psi)n}$. Each window is of length $n/\Psi$, and hence these two window secrets cover all $(\Psi + 1)n/\Psi$ elements in superwindow $W_{\ell n}$. Using the random nonce $r_i$ in each share $S_i$, the legitimate receiver can calculate $\kappa_i$ by hashing $r_i$ with the appropriate window secret.